```
105         p.sellingPrice = read_double("Enter selling price: ");
106         p.quantity = read_integer("Enter quantity: ");
107         add(store.products, p);  // Use add to add product
108     }
109 }
110 
111 // Print all product information
112 void print_products(const StoreData &store)
113 {
114     printf("Current Products:\n");
115     for (int i = 0; i < store.product_count; i++)
116     {
117         Product p = get(store.products, i);
118         printf("Index %d: Name: %s, Cost Price: %.2f, Selling Price: %.2f, Quantity:
        ↪  %d\n",
119                i, p.name.c_str(), p.costPrice, p.sellingPrice, p.quantity);
120     }
121 }
122 
123 // Add new product
124 void add_product(StoreData &store)
125 {
126     Product p;
127     p.name = read_string("Enter product name: ");
128     p.costPrice = read_double("Enter cost price: ");
129     p.sellingPrice = read_double("Enter selling price: ");
130     p.quantity = read_integer("Enter quantity: ");
131 
132     if (add(store.products, p))  // Use add to add product
133     {
134         store.product_count++;
135         printf("Product added successfully.\n");
136     }
137     else
138     {
139         printf("Failed to add product.\n");
140     }
141 }
142 
143 // Remove product
144 void remove_product(StoreData &store)
145 {
146     print_products(store);
147     printf("Please enter an index to remove: ");
148     int index = read_integer("Index:");
149 
150     if (index < 0 || index >= store.product_count)
151     {
152         printf("Invalid index. No product removed.\n");
153         return;
154     }
155 
156     // Use resize logic to remove element - shift elements forward
157     for (int i = index; i < store.product_count - 1; i++)
```

```
158     {
159         Product next = get(store.products, i + 1);
160         set(store.products, i, next);
161     }
162 
163     // Adjust array size
164     if (!resize(store.products, store.products->capacity))
165     {
166         printf("Error resizing array after removal.\n");
167     }
168 
169     store.product_count--;
170     printf("Product removed successfully.\n");
171 }
172 
173 // Print update menu
174 void print_update_menu()
175 {
176     printf("Update Menu:\n");
177     printf("1. Update name\n");
178     printf("2. Update cost\n");
179     printf("3. Update the price\n");
180     printf("4. Update number on hand\n");
181     printf("5. Quit update\n");
182 }
183 
184 // Update product information
185 void update_products(StoreData &store)
186 {
187     print_products(store);
188     int index = read_integer("Enter product index to update: ");
189 
190     if (index < 0 || index >= store.product_count)
191     {
192         printf("Invalid index.\n");
193         return;
194     }
195 
196     int update_option;
197     do
198     {
199         print_update_menu();
200         update_option = read_integer("Enter update option: ");
201 
202         Product p = get(store.products, index);  // Get current product
203 
204         switch (update_option)
205         {
206         case 1:
207             p.name = read_string("Enter new name: ");
208             set(store.products, index, p);  // Use set to update product
209             break;
210         case 2:
211             p.costPrice = read_double("Enter new cost price: ");
```

```
            set(store.products, index, p);
            break;
        case 3:
            p.sellingPrice = read_double("Enter new selling price: ");
            set(store.products, index, p);
            break;
        case 4:
            p.quantity = read_integer("Enter new quantity: ");
            set(store.products, index, p);
            break;
        case 5:
            printf("Quitting update.\n");
            break;
        default:
            printf("Invalid option.\n");
            break;
        }
    } while (update_option != 5);
}

// Print store summary
void print_store_summary(const StoreData &store)
{
    printf("Store Summary:\n");
    printf("Total Products: %d\n", store.product_count);
    printf("Total Sales Value: %.2f\n", calculate_sale_value(store));
    printf("Total Profit: %.2f\n", calculate_profit(store));
    printf("Total Store Value: %.2f\n", calculate_store_value(store));
    printf("Low Stock Items: %d\n", count_low_stock(store));
}

// Print main menu
void print_menu()
{
    printf("\nMain Menu:\n");
    printf("1. Add Product\n");
    printf("2. Remove Product\n");
    printf("3. Update Product\n");
    printf("4. Print Store Summary\n");
    printf("5. Exit\n");
}

// Main function
int main()
{
    StoreData my_store = {0, 0.0, 0.0, nullptr}; // Initialize store data
    my_store.products = dynamic_array_create<Product>(); // Create dynamic array

    if (my_store.products == nullptr)
    {
        printf("Error: Failed to create dynamic array for products.\n");
        return 1;
    }

```

```
266     populate_array(my_store); // Initialize product array
267 
268     int option;
269     do
270     {
271         print_menu();
272         option = read_integer("Enter your choice: ");
273 
274         switch (option)
275         {
276         case ADD_PRODUCT:
277             add_product(my_store);
278             break;
279         case REMOVE_PRODUCT:
280             remove_product(my_store);
281             break;
282         case UPDATE_PRODUCT:
283             update_products(my_store);
284             break;
285         case PRINT_SUMMARY:
286             print_store_summary(my_store);
287             break;
288         case EXIT:
289             printf("Exiting program.\n");
290             break;
291         default:
292             printf("Invalid choice. Please try again.\n");
293             break;
294         }
295     } while (option != EXIT);
296 
297     // Free dynamic array memory
298     if (my_store.products != nullptr)
299     {
300         delete_dynamic_array(my_store.products);
301     }
302 
303     return 0;
304 }
```

This file has additional line breaks applied by OnTrack because they contain lines longer than the configured limit. Lines over 1000 characters long have been truncated to limit PDF page count. The orginal submission can be retrieved via the "Download Uploaded Files" function.

```cpp
#include <stdlib.h>
#include <cstdio>

/**
 * Node structure for linked list
 */
template <typename T>
struct Node
{
    Node<T>* next;
    T data;
};

/**
 * Linked list structure
 */
template <typename T>
struct LinkedList
{
    Node<T>* first;
    Node<T>* last;
    int size;
};

/**
 * Menu options enum
 */
enum MenuOption
{
    PRINT_LIST = 1,
    ADD_NODE = 2,
    INSERT_AT_INDEX = 3,
    DELETE_AT_INDEX = 4,
    EXIT_PROGRAM = 5
};

/**
 * Creates a new linked list
 */
template <typename T>
LinkedList<T>* create_linked_list()
{
    LinkedList<T>* list = (LinkedList<T>*)malloc(sizeof(LinkedList<T>));
    list->first = nullptr;
    list->last = nullptr;
    list->size = 0;
    return list;
}

/**
```

```cpp
 * Add node to the end of the list
 */
template <typename T>
void add_node(LinkedList<T>* list, T data)
{
    Node<T>* new_node = (Node<T>*)malloc(sizeof(Node<T>));
    new_node->data = data;
    new_node->next = nullptr;

    if (list->first == nullptr)
    {
        list->first = new_node;
        list->last = new_node;
    }
    else
    {
        list->last->next = new_node;
        list->last = new_node;
    }
    list->size++;
}

/**
 * Print linked list
 */
template <typename T>
void print_linked_list(LinkedList<T>* list)
{
    if (list->first == nullptr)
    {
        printf("List is empty!\n");
        return;
    }

    printf("Linked List (%d nodes): ", list->size);
    Node<T>* current = list->first;

    while (current != nullptr)
    {
        printf("%d", current->data);
        if (current->next != nullptr)
        {
            printf(" -> ");
        }
        current = current->next;
    }
    printf(" -> NULL\n");
}

/**
 * Insert node at specific index
 */
template <typename T>
void insert_node_at_index(LinkedList<T>* list, int index, T data)
```

```
105 {
106     if (index < 0 || index > list->size)
107     {
108         printf("Error: Index %d out of bounds! List size: %d\n", index, list->size);
109         return;
110     }
111 
112     Node<T>* new_node = (Node<T>*)malloc(sizeof(Node<T>));
113     new_node->data = data;
114 
115     if (index == 0)
116     {
117         new_node->next = list->first;
118         list->first = new_node;
119         if (list->last == nullptr)
120         {
121             list->last = new_node;
122         }
123     }
124     else if (index == list->size)
125     {
126         new_node->next = nullptr;
127         list->last->next = new_node;
128         list->last = new_node;
129     }
130     else
131     {
132         Node<T>* current = list->first;
133         for (int i = 0; i < index - 1; i++)
134         {
135             current = current->next;
136         }
137         new_node->next = current->next;
138         current->next = new_node;
139     }
140     list->size++;
141     printf("Node with value %d inserted at index %d\n", data, index);
142 }
143 
144 /**
145  * Delete node at specific index
146  */
147 template <typename T>
148 void delete_node_at_index(LinkedList<T>* list, int index)
149 {
150     if (index < 0 || index >= list->size)
151     {
152         printf("Error: Index %d out of bounds! List size: %d\n", index, list->size);
153         return;
154     }
155 
156     Node<T>* node_to_delete;
157 
158     if (index == 0)
```

```
159     {
160         node_to_delete = list->first;
161         list->first = list->first->next;
162         if (list->last == node_to_delete)
163         {
164             list->last = nullptr;
165         }
166     }
167     else
168     {
169         Node<T>* current = list->first;
170         for (int i = 0; i < index - 1; i++)
171         {
172             current = current->next;
173         }
174         node_to_delete = current->next;
175         current->next = node_to_delete->next;
176 
177         if (list->last == node_to_delete)
178         {
179             list->last = current;
180         }
181     }
182 
183     printf("Node with value %d deleted from index %d\n", node_to_delete->data,
     ↪  index);
184     free(node_to_delete);
185     list->size--;
186 }
187 
188 /**
189  * Delete entire linked list
190  */
191 template <typename T>
192 void delete_linked_list(LinkedList<T>* list)
193 {
194     Node<T>* current = list->first;
195     Node<T>* next;
196 
197     while (current != nullptr)
198     {
199         next = current->next;
200         free(current);
201         current = next;
202     }
203 
204     free(list);
205 }
206 
207 /**
208  * Print main menu
209  */
210 void print_menu()
211 {
```

```
212     printf("\nLinked List Operations:\n");
213     printf("%d. Print Linked List\n", PRINT_LIST);
214     printf("%d. Add Node to End\n", ADD_NODE);
215     printf("%d. Insert Node at Index\n", INSERT_AT_INDEX);
216     printf("%d. Delete Node at Index\n", DELETE_AT_INDEX);
217     printf("%d. Exit\n", EXIT_PROGRAM);
218 }
219 
220 /**
221  * Main function
222  */
223 int main()
224 {
225     LinkedList<int>* list = create_linked_list<int>();
226     int choice;
227 
228 
229     while (true)
230     {
231         print_menu();
232         printf("Enter your choice (1-5): ");
233         scanf("%d", &choice);
234 
235         switch (choice)
236         {
237             case PRINT_LIST:
238                 print_linked_list(list);
239                 break;
240 
241             case ADD_NODE:
242                 {
243                     int value;
244                     printf("Enter value to add: ");
245                     scanf("%d", &value);
246                     add_node(list, value);
247                     printf("Node added to end of list\n");
248                 }
249                 break;
250 
251             case INSERT_AT_INDEX:
252                 {
253                     int index, value;
254                     printf("Enter index to insert at: ");
255                     scanf("%d", &index);
256                     printf("Enter value to insert: ");
257                     scanf("%d", &value);
258                     insert_node_at_index(list, index, value);
259                 }
260                 break;
261 
262             case DELETE_AT_INDEX:
263                 {
264                     int index;
265                     printf("Enter index to delete: ");
```

```
266                     scanf("%d", &index);
267                     delete_node_at_index(list, index);
268                 }
269                 break;
270 
271             case EXIT_PROGRAM:
272                 delete_linked_list(list);
273                 printf("Program exited.\n");
274                 return 0;
275 
276             default:
277                 printf("Invalid choice! Please enter 1-5.\n");
278                 break;
279         }
280     }
281 
282     return 0;
283 }
```

15 Custom Program - Pitch

A key element of the Distinction grade is the ability to apply your understanding in new contexts. To achieve this, you need to design and build your own program using the knowledge and skills that you have developed during this unit.This is the first of a number of tasks associated with building your own custom program. Use this task to describe what you are planning on building, we can then give you feedback to help ensure you are on track for your target grade.

Outcome	Description
TLO1	You can demonstrate the ability to scope and pitch a project

Date	Author	Comment
2025/08/06 20:24	Ziyue Meng	Planned date adjusted to 12 Sep.
2025/10/08 18:09	Ziyue Meng	Ready for Feedback
2025/10/11 08:03	Andrew Cain	This looks like an interesting project that should have scope for you to demonstrate the different programming skills that you have developed in the unit, and to grow into something that will meet the D/HD criteria. Make sure to focus on having a good model in your program (as you learnt in the programming project credit task) and adding in features that get the program to do things.
2025/10/11 08:03	Andrew Cain	Remember this is not about machine learning, but creating a software model within your code and getting that to do things...
2025/10/11 08:04	Andrew Cain	You should be able to iterate on your project so that you can build it up to the required standard. Attend classes and show your progress to the staff to get feedback and advice, and ideas on how to structure or expand your project if needed.
2025/10/11 08:04	Andrew Cain	Discuss
2025/10/11 08:04	Andrew Cain	The staff in your applied can sign this off now. Make sure to attend the class to discuss your project, and show them where you are up to with it. You want to include them in the process so that they can help make sure you are on track for success.
2025/10/16 18:17	Ziyue Meng	Dr., I believe my code does not primarily use machine learning; it only serves as an auxiliary function. The main focus of my code is on processing large volumes of CSV data and creating visualizations to make the data more understandable, thereby helping people make simpler decisions when they have travel needs.
2025/10/16 20:07	Andrew Cain	Sounds good. Make sure to discuss in your applied so they are aware and will sign this off
2025/10/21 15:44	Ganesh Krishnasamy	Complete

Monash

Introduction to Programming

OnTrack Submission

Custom Program - Pitch

Submitted By:
Ziyue Meng
zmen0034
2025/10/08 18:09

Tutor:
Andrew Cain

Task Outcomes		Supports
TLO1	You can demonstrate the ability to scope and pitch a project that brings together the skills you have developed in the unit.	ULO1 Use Instructions ULO2 Programs, Functions, and Types ULO3 Use libraries ULO4 Use good programming practices

October 8, 2025

Custom Project

Name: ZIyue Meng
Student ID: 36035432
Date: 5/10/2025

Project Pitch (D2)

Programming Language(s): Python
Using GenAI: Copilot
Target Grade:HD

Overview:

This project develops an intelligent travel planning assistant that provides personalized destination recommendations using machine learning algorithms. The system processes a comprehensive dataset of 100 international travel destinations collected from online sources, offering users a wide selection of global travel options.

The application features a user-friendly graphical interface with three main components: a destination browser for exploring travel locations, a recommendation system that suggests destinations based on user preferences, and basic data visualization for understanding recommendation results. Users can set their travel preferences including destination type, budget level, and preferred regions to receive customized suggestions.

The core functionality is built using Python's scikit-learn library, implementing K-Nearest Neighbors algorithm for finding similar destinations and Random Forest regression for predicting user ratings. The system handles data management through pandas and presents results through an intuitive Tkinter-based interface.

HD Extension Ideas:

- Automated Data Integration
  - Implement web scraping to dynamically collect and update destination data from travel websites
  - Develop real-time data updating system with automated data validation
  - Integrate multiple online data sources for comprehensive destination information
- Advanced Machine Learning Features
  - Implement collaborative filtering using matrix factorization for improved recommendations
  - Develop content-based filtering that analyzes destination features and user preferences

    - Create hybrid recommendation system combining multiple ML approaches
    - Implement model performance evaluation and comparison framework
- Enhanced Visualization System
    - Develop interactive data dashboards with multiple chart types
    - Implement geospatial visualization showing destinations on maps
    - Create comparative analysis charts for destination features and ratings
    - Build real-time visualization updates based on user interactions
- Intelligent Recommendation Features
    - Implement recommendation explanation system showing why destinations are suggested
    - Develop multi-criteria recommendation considering season, duration, and activities
    - Create similarity analysis showing how destinations relate to each other
    - Build preference learning system that adapts based on user feedback
- Advanced User Features
    - Implement user profile management with preference history
    - Develop saved recommendations and travel planning functionality
    - Create destination comparison tools with side-by-side feature analysis
    - Build export functionality for recommendations and travel plans
- System Optimization
    - Implement caching system for faster recommendation generation
    - Develop automated data preprocessing and cleaning pipelines
    - Create performance monitoring and optimization features
    - Build modular architecture for easy feature expansion
-

How did you come up with your project pitch?

This project was inspired by the challenge of planning travel in an increasingly complex world with countless destination options. Traditional travel planning often involves sifting through overwhelming amounts of information without personalized guidance. I identified an opportunity to apply machine learning techniques to create a smart recommendation system that understands individual preferences.

The concept combines practical travel needs with data science applications, addressing the real problem of destination choice overload. By focusing on

personalized recommendations, the system helps users discover destinations they might otherwise overlook. The use of a comprehensive dataset from online sources ensures the recommendations are based on extensive, real-world travel information.

The HD extensions were designed to push the project beyond basic functionality into a sophisticated travel intelligence platform. These enhancements focus on improving recommendation accuracy, providing deeper insights through advanced visualization, and creating a more engaging user experience through interactive features. The project demonstrates how machine learning can transform everyday tasks like travel planning into intelligent, data-driven experiences.

Project Progress (D3)

Prior Knowledge and Experience

Iterations and Study to Create Project and Extend Capabilities

To build this project, and extend my programming capabilities, I did the following:

<date> - <describe action>
<provide a screenshot image or notes of what **you did/created** related to this>
<describe what you learnt from this>

Project Wrap Up (D4)

Progress in Wrapping Up the Project

To build this project, and extend my programming capabilities, I did the following:

<date> - <describe action>
<provide a screenshot image or notes of what **you did/created** related to this>
<describe what you learnt from this>

Distinction Achievement

Impressive - high quality code

Functional and Level of Completion

Data Organisation

Functional Decomposition - use of functions, procedures, methods

Build and Run Instructions

Demo Video Link

High Distinction Achievement (H2)

Other Libraries / Language Features / Tools Used

Degree of Ownership and Future Direction

Efficient Core Architecture

Creativity and sophistication

References

Generative AI acknowledgement

16 Programming Project

Now that you have all of the foundational programming knowledge and skills, this task gets you to put these together using processes that will help you build larger and more interesting projects. The focus of this task is more on understanding the process, and how to approach building larger projects. This will include looking at test driven development tools and seeing how you can use them to continuously check that your code is working as you intended.

Date	Author	Comment
2025/08/06 20:24	Ziyue Meng	Planned date adjusted to 19 Sep.
2025/10/07 02:37	Ziyue Meng	Working On It
2025/10/09 01:02	Ziyue Meng	Ready for Feedback
2025/10/12 14:48	Ganesh Krishnasamy	please come and see me during the applied/workshop session.
2025/10/12 14:49	Ganesh Krishnasamy	Discuss
2025/10/21 15:58	Ganesh Krishnasamy	1) You need to throw an error whenever the customer orders something that does not exist in the stock of the shop. 2) There is an infinite loop the order model.
2025/10/21 15:58	Ganesh Krishnasamy	Fix and Resubmit
2025/10/21 16:01	Ganesh Krishnasamy	Discuss
2025/10/22 16:54	Ganesh Krishnasamy	discussed. all works now.
2025/10/22 16:54	Ganesh Krishnasamy	Complete

Monash

Introduction to Programming

OnTrack Submission

Programming Project

Submitted By:
Ziyue Meng
zmen0034
2025/10/09 01:02

Tutor:
Ganesh Krishnasamy

October 9, 2025

Programming Project

Name: Ziyue Meng
Student ID: 36035432
Date: 8/10/2025

Learning Journey and Evidence

Prior Knowledge and Experience

I have a solid foundation in several key aspects of programming. I've mastered the use of pointers and references. Pointers are variables that store memory addresses, which allow for direct memory manipulation and efficient data passing between functions. References, on the other hand, act as an alias for an existing variable, providing a more intuitive way to access and modify data in some cases.

I'm also well - versed in loops. For example, the for loop is useful when you know exactly how many times you want to iterate. It has a concise syntax where you can initialize a counter, set a condition for continuation, and update the counter in each iteration. The while loop is more flexible, as it continues as long as a given condition is true, making it suitable for situations where the number of iterations is not fixed in advance. The do - while loop is similar to the while loop, but it executes the loop body at least once before checking the condition.

Enumerations (enum) are a user - defined data type in programming. They allow you to create a set of named constants. For instance, if you are creating a program about the days of the week, you can define an enum like enum Days {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY}; which makes the code more readable and maintainable.

Regarding structures, I can define a structure to encapsulate related data. For example, a "Student" struct containing name, ID, and score. I'm able to properly access its members, which is crucial for handling complex data entities in programming.

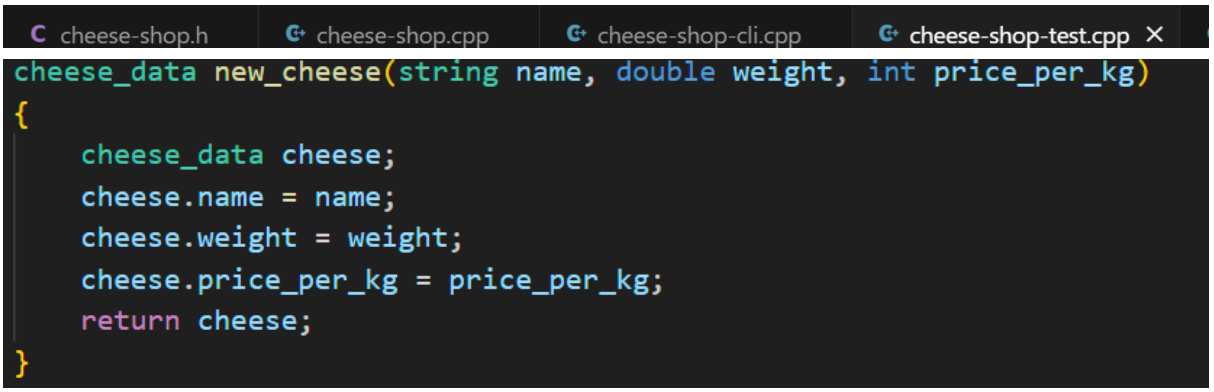
When it comes to arrays, I have a good grasp of their usage. I know how to declare and initialize both one - dimensional and two - dimensional arrays. For a one - dimensional array, it can be done like int numbers[5] = {1, 2, 3, 4, 5}; and for a two - dimensional array, something like int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}} works. I'm familiar with iterating through array elements using loops, which is essential for processing the data they hold. Arrays are great for storing collections of similar data, such as a list of student scores or a series of temperatures. I also understand the importance of being cautious with array index bounds to avoid accessing elements outside the valid range, which can lead to unexpected program behavior.

Study to Acquire, Refresh, and/or Extend Capability

To study and master this topic I did the following:

Setting up the project

1. Create a folder for your project.
2. Download the testing framework code:
    - catch_amalgamated.hpp
    - catch_amalgamated.cpp
3. Create the following files:
    - **cheese-shop.h**: a header file for our shared functions and data types.
    - **cheese-shop-test.cpp**: the code for our tests.
    - **cheese-shop.cpp**: the code for the model of our program.
    - **cheese-shop-cli.cpp**: the command line interface (CLI) code.



```
8  TEST_CASE("Customer Order - Create new customer order")
9  {
10     customer_order order = new_customer_order("John Doe");
11
12     REQUIRE(order.order_id > 0);
13     REQUIRE(order.customer_name == "John Doe");
14     REQUIRE(order.cheeses.size() == 0);
15     REQUIRE(order.fulfilled == false);
16 }
17
18 TEST_CASE("Customer Order - Add cheese to order")
19 {
20     customer_order order = new_customer_order("Jane Smith");
21
22     cheese_data cheese1 = new_cheese("Cheddar", 1.5, 2000);
23     cheese_data cheese2 = new_cheese("Gouda", 0.5, 1500);
24
25     add_cheese_to_order(order, cheese1);
26     add_cheese_to_order(order, cheese2);
27
28     REQUIRE(order.cheeses.size() == 2);
29     REQUIRE(order.cheeses[0].name == "Cheddar");
30     REQUIRE(order.cheeses[0].weight == Approx(1.5));
31     REQUIRE(order.cheeses[1].name == "Gouda");
32     REQUIRE(order.cheeses[1].weight == Approx(0.5));
33 }
34
```

This was my first exposure to Test-Driven Development, starting with creating simple test cases and learning how to define expected behavior before implementing functionality. Although the initial compilation failures were confusing at first, I quickly understood this was a normal part of the TDD process—the "red light" phase in the red-green-refactor cycle. By implementing the basic `new_cheese` function, I grasped fundamental C++ concepts like header guards, default parameters, and struct definitions, laying a solid foundation for subsequent development.

Add in additional cheese functionality

Now you can add unit tests and the code to make the following changes.

1. Add a `total_cost` function that accepts a `cheese_data` value and returns the cost (in cents) of that cheese. Test to ensure this works with different values, and include this in the output line.
2. Extend the details in the cheese string to include the total cost, so that this outputs the total price of the cheese in addition to the name, weight, and price per kilogram.
3. Create a `reduce_weight` procedure that accepts a reference to a `cheese_data` value, and is given a weight to reduce the cheese by. It should ignore negative values, and ensure that the final weight does not become negative. This will return the actual amount removed. For example, if you have 1.5 kg of cheese and attempt to reduce it by 1.7 kg then the resulting weight will be 0, and 1.5 will be returned. This will be used when orders are fulfilled in later iterations - you can add basic test code to the CLI to see this functioning.
4. Create a `increase_weight` procedure that accepts a reference to a `cheese_data` value, and is given a weight to add to the cheese. This should ignore negative values, but otherwise add the passed in weight to the weight already present. This will be used when stock is re-supplied in later iterations - you can add basic test code to the CLI to see this functioning.

For each of these, first develop the required unit tests and then add the code to the model and add something to the CLI version to see these working yourself.

```
18  string cheese_to_string(const cheese_data &cheese, bool full_details)
19  {
20      if (full_details)
21      {
22          double total_price = total_cost(cheese) / 100.0;
23          return format("{}: {:.2f} kg, ${:.2f} per kg, ${:.2f} total",
24                        cheese.name,
25                        cheese.weight,
26                        cheese.price_per_kg / 100.0,
27                        total_price);
28      }
29      else
30      {
31          return cheese.name;
32      }
33  }
```

```
40  double reduce_weight(cheese_data &cheese, double amount)
41  {
42      if (amount <= 0)
43          return 0.0;
44
45      double actual_removed = amount;
46      if (amount > cheese.weight)
47      {
48          actual_removed = cheese.weight;
49      }
50
51      cheese.weight -= actual_removed;
52      return actual_removed;
53  }
```

```cpp
double reduce_weight(cheese_data &cheese, double amount)
{
    if (amount <= 0)
        return 0.0;

    double actual_removed = amount;
    if (amount > cheese.weight)
    {
        actual_removed = cheese.weight;
    }

    cheese.weight -= actual_removed;
    return actual_removed;
}

void increase_weight(cheese_data &cheese, double amount)
{
    if (amount > 0)
    {
        cheese.weight += amount;
    }
}
```

During this phase, I delved deeper into designing comprehensive data models. By implementing various operation functions for cheese data, I understood the importance of encapsulating business logic. The implementation of the `cheese_to_string` function particularly taught me techniques for string formatting and floating-point precision control. Simultaneously, developing the habit of writing test cases for each new feature helped me appreciate how TDD ensures code quality and functional correctness.

```cpp
#include <string>
#include <vector> // or your dynamic array header

using std::string;
using std::vector; // unless using your dynamic array

/**
 * Data about a cheese within an order or in stock.
 *
 * @field name The name of the cheese.
 * @field weight The weight of the cheese in stock (kg).
 * @field price_per_kg The price of the cheese per kg (cents).
 */
struct cheese_data
{
    string name;
    double weight;
    int price_per_kg;
};

/**
 * Data about the cheese shop - the stock on hand.
 *
 * @field cheeses The list of cheeses in stock.
 */
struct shop_data
{
    vector<cheese_data> cheeses; // or your dynamic array
};

//...
```

```cpp
47  struct shop_data
48  {
49      vector<cheese_data> cheeses;
50      vector<customer_order> orders;
51      int next_order_id;
52  };
53
```

6. Now you can add the `handle_add_cheese` procedure. This accepts the shop and uses your `read_cheese` to get the data, and then pass this to `add_cheese` to add the new cheese into the shop.

7. In order to see that this has worked, we need to add the `print_stock_list` procedure. This should output a report listing all the stock within the shop. Add a header and footer to the report to make sure that it is easy to see where it starts and ends.

8. With these in place you can now update `main`. Remove the cheese, and put in a shop. Initially this can just be some small test code, so you can see it work.

9. Finish up by creating a `read_main_menu`. This can output options for 0: Quit, 1: Add Cheese, and 2: Print Stock List - matching the values in the `main_menu_option`. Then update main to read the user's choice into a variable, loop while they do not choose to quit, either adding cheese (with `handle_add_cheese`) or printing the stock list.

   Run your program, and you should see we now have a functional component.

```
155 void add_customer_order(shop_data &shop, const customer_order &order)
156 {
157     shop.orders.push_back(order);
158 }
159
160 void remove_customer_order(shop_data &shop, int index)
161 {
162     if (index >= 0 && index < shop.orders.size())
163     {
164         shop.orders.erase(shop.orders.begin() + index);
165     }
166 }
167
```

Transitioning from single entity management to collection management represented significant progress. I learned to use STL vectors for managing dynamic collections and designed comprehensive inventory data structures. When first attempting to separate model from interface, I encountered function call order issues, which were resolved by adding forward declarations. This phase taught me the importance of good architectural design for code maintainability.

```
11  /**
12   * The list of options in the main menu.
13   */
14  enum main_menu_option
15  {
16      EXIT_MAIN_MENU,
17      ADD_CHEESE_MENU,
18      EDIT_CHEESE_MENU,
19      DELETE_CHEESE_MENU,
20      PRINT_STOCK_LIST_MENU,
21      ADD_CUSTOMER_ORDER_MENU,
22      FULFILL_ORDER_MENU,
23      PRINT_ORDERS_MENU
24  };
25
```

```
329 /**
330  * Show the main menu and get the chosen option from the user.
331  */
332 main_menu_option read_main_menu_option()
333 {
334     write_line("\n=== Cheese Shop Main Menu ===");
335     write_line("0. Exit");
336     write_line("1. Add cheese");
337     write_line("2. Edit cheese");
338     write_line("3. Delete cheese");
339     write_line("4. Print stock list");
340     write_line("5. Add customer order");
341     write_line("6. Fulfill order");
342     write_line("7. Print orders");
343     return static_cast<main_menu_option>(read_integer("Select an option (0-7): ", 0, 7));
344 }
345
```

Developing a complete command-line interface allowed me to focus on user experience design. By implementing secure input validation functions, I learned to handle various edge cases and user input errors. Using enum types significantly improved code readability, while modular function design made the menu system easy to extend and maintain. This phase made me realize that a good user interface requires not only complete functionality but also robust error handling.

```
/**
 * Customer order containing multiple cheese items.
 *
 * @field order_id Unique identifier for the order
 * @field customer_name Name of the customer
 * @field cheeses List of cheeses in the order
 * @field fulfilled Whether the order has been fulfilled
 */
struct customer_order
{
    int order_id;
    string customer_name;
    vector<cheese_data> cheeses;
    bool fulfilled;
};
```

Implementing the customer order system was the most challenging part of the entire project. I needed to design complex relationships between multiple entities and ensure data consistency. The linkage logic between order fulfillment and inventory was particularly complex, requiring careful handling of various edge cases. Through this functionality, I deeply understood the complexity of business logic and how to maintain data integrity in code. Comprehensive test coverage helped me discover many potential issues.

```
ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/c3work
$
      |                       remove_cheese
./cheese-shop.h:63:6: note: 'remove_cheese' declared here
   63 | void remove_cheese(shop_data &shop, int index);
      |      ^
cheese-shop-cli.cpp:65:5: error: unknown type name 'order_data'
   65 |     order_data order;
      |     ^
3 errors generated.

      |                       remove_cheese
./cheese-shop.h:63:6: note: 'remove_cheese' declared here
   63 | void remove_cheese(shop_data &shop, int index);
      |      ^
cheese-shop-cli.cpp:65:5: error: unknown type name 'order_data'
   65 |     order_data order;
      |     ^
      |                       remove_cheese
./cheese-shop.h:63:6: note: 'remove_cheese' declared here
   63 | void remove_cheese(shop_data &shop, int index);
      |      ^
cheese-shop-cli.cpp:65:5: error: unknown type name 'order_data'
      |                       remove_cheese
./cheese-shop.h:63:6: note: 'remove_cheese' declared here
   63 | void remove_cheese(shop_data &shop, int index);
      |      ^
      |                       remove_cheese
./cheese-shop.h:63:6: note: 'remove_cheese' declared here
   63 | void remove_cheese(shop_data &shop, int index);
      |                       remove_cheese
      |                       remove_cheese
./cheese-shop.h:63:6: note: 'remove_cheese' declared here
   63 | void remove_cheese(shop_data &shop, int index);
      |      ^
cheese-shop-cli.cpp:65:5: error: unknown type name 'order_data'
   65 |     order_data order;
      |     ^
0 ⚠ 0
```

After completing the basic functionality, I shifted focus to code quality and user experience optimization. By refactoring duplicate code and improving output formats, I learned how to make code clearer and more readable. This phase made me realize that software development is not just about implementing features, but more importantly about creating systems that are easy to maintain and use. Good code structure and clear user interfaces are equally important.

cheese-shop.cpp > cheese_to_string(const cheese_data &, bool)

```cpp
#include "cheese-shop.h"
#include <format>

using std::format;

cheese_data new_cheese(string name, double weight, int price_per_kg)
{
    cheese_data cheese;
    cheese.name = name;
    cheese.weight = weight;
    cheese.price_per_kg = price_per_kg;
    return cheese;
}

string cheese_to_string(const cheese_data &cheese, bool full_details)
{
    if (full_details)
    {
        return format("{}: {:.2f} kg, ${:.2f}",
            cheese.name,
            cheese.weight,
            cheese.price_per_kg / 100.0);
    }
    else
    {
        return cheese.name;
    }
}
```

The actual deployment process presented numerous technical challenges, particularly with C++20's `std::format` support across different compilation environments. By researching alternative solutions and adjusting compilation configurations, I learned how to solve practical engineering problems. This experience taught me that theoretical knowledge must be combined with practical environments, and the ability to solve problems flexibly is a crucial quality for software developers.

OUTPUT:

```
ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/c3work
$ ./cheese-shop.exe
Welcome to the Cheese Shop!

=== Cheese Shop Main Menu ===
0. Exit
1. Add cheese
2. Edit cheese
3. Delete cheese
4. Print stock list
5. Add customer order
6. Fulfill order
7. Print orders
Select an option (0-7): 1
Enter cheese name: cheddar
Enter weight in stock (kg): 5.0
Enter price per kg (cents): 1500
Cheese added successfully!

=== Cheese Shop Main Menu ===
0. Exit
1. Add cheese
2. Edit cheese
3. Delete cheese
4. Print stock list
5. Add customer order
6. Fulfill order
7. Print orders
Select an option (0-7): 4

==========================
Cheese stock list:
==========================
cheddar: 5.00 kg, $15.00 per kg, $75.00 total
==========================
```

```
=== Cheese Shop Main Menu ===
0. Exit
1. Add cheese
2. Edit cheese
3. Delete cheese
4. Print stock list
5. Add customer order
6. Fulfill order
7. Print orders
Select an option (0-7): 5
Enter customer name: Ziyue
Adding cheeses to order. Enter empty cheese name to finish.
Enter cheese name: cheddar
Enter weight needed (kg): 1.5
Enter cheese name:
Enter weight needed (kg):
Invalid input. Please enter a valid number.
Enter weight needed (kg):
Invalid input. Please enter a valid number.
Enter weight needed (kg):
Invalid input. Please enter a valid number.
Enter weight needed (kg):
Invalid input. Please enter a valid number.
Enter weight needed (kg): 1
Order #1 created successfully with 1 cheese items.

=== Cheese Shop Main Menu ===
0. Exit
1. Add cheese
2. Edit cheese
3. Delete cheese
4. Print stock list
5. Add customer order
6. Fulfill order
7. Print orders
Select an option (0-7): 7
```

```
=== Customer Orders ===
Order #1 - Customer: Ziyue
Status: Pending
Cheeses:
  1: cheddar: 1.50 kg, $0.00 per kg, $0.00 total
Total cost: $0.00
---

=== Cheese Shop Main Menu ===
0. Exit
1. Add cheese
2. Edit cheese
3. Delete cheese
4. Print stock list
5. Add customer order
6. Fulfill order
7. Print orders
Select an option (0-7): 6
Select an order:
1: Order #1 - Ziyue (Pending)
Select order (0 to cancel): 1
Order #1 fulfilled successfully!
Total amount: $22.50

=== Cheese Shop Main Menu ===
0. Exit
1. Add cheese
2. Edit cheese
3. Delete cheese
4. Print stock list
5. Add customer order
6. Fulfill order
7. Print orders
Select an option (0-7): 4

=========================
Cheese stock list:
=========================
cheddar: 3.50 kg, $15.00 per kg, $52.50 total
=========================
```

```
=== Customer Orders ===
Order #1 - Customer: Ziyue
Status: Pending
Cheeses:
  1: cheddar: 1.50 kg, $0.00 per kg, $0.00 total
Total cost: $0.00
---

=== Cheese Shop Main Menu ===
0. Exit
1. Add cheese
2. Edit cheese
3. Delete cheese
4. Print stock list
5. Add customer order
6. Fulfill order
7. Print orders
Select an option (0-7): 6
Select an order:
1: Order #1 - Ziyue (Pending)
Select order (0 to cancel): 1
Order #1 fulfilled successfully!
Total amount: $22.50

=== Cheese Shop Main Menu ===
0. Exit
1. Add cheese
2. Edit cheese
3. Delete cheese
4. Print stock list
5. Add customer order
6. Fulfill order
7. Print orders
Select an option (0-7): 4

=========================
Cheese stock list:
=========================
cheddar: 3.50 kg, $15.00 per kg, $52.50 total
=========================
```

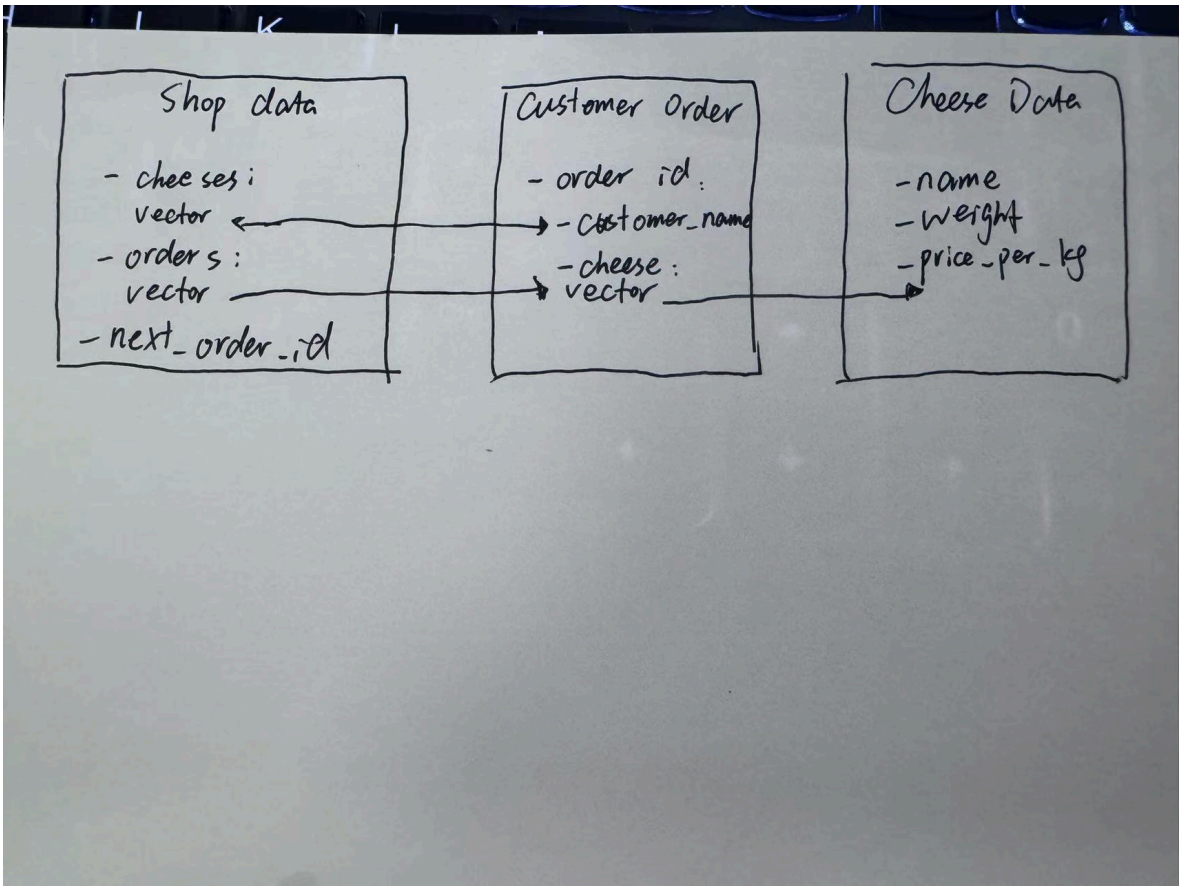
```
ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/c3work
$ ./test.exe
Randomness seeded to: 3638091055
===============================================================================
All tests passed (30 assertions in 8 test cases)
```

Through the complete development process of this cheese shop project, I not only mastered the full workflow of Test-Driven Development but also gained a deep understanding of the software development lifecycle—from requirements analysis to code implementation, and finally to problem resolution. Each phase brought new challenges and learning opportunities, from initial basic syntax learning to final system integration, with each step being a valuable growth experience. Most importantly, I learned how to maintain patience when encountering problems and analyze and solve them through systematic approaches—this ability will have a profound impact on my future programming learning journey.

Brief Summary of Concepts

Concept / Commands	Key Idea / Concept
Unit Test	A test that verifies the correctness of individual code units in isolation from other components.
Test Drive Development	A development methodology where tests are written before implementation code, following the red-green-refactor cycle.
MVC	Model-View-Controller architecture pattern that separates data, presentation, and control logic for better maintainability.
Iterative Development	A development approach that builds software incrementally through repeated cycles of planning, implementation, and evaluation.

Design for Cheese Shop Extension



The extended cheese shop system now organizes data around three core entities. The Shop Data acts as the central container, managing both inventory (`cheeses`) and customer transactions (`orders`). Each Customer Order represents a single transaction, containing the customer's details and their selected cheeses. The Cheese Data structure serves as the fundamental unit, reused across both inventory management and order processing.

The key innovation in this design is the bidirectional relationship between orders and inventory. When orders are fulfilled, the system automatically reduces stock levels, maintaining real-time inventory accuracy. This integrated approach ensures data consistency while supporting the complete business workflow from order creation to fulfillment.

Reflection

What is the most important thing you learned from this and why?

The most important thing I learned from this project is the practical value of Test Driven Development in creating reliable and maintainable code. By writing tests first, I was forced to think carefully about design decisions and expected behaviors before implementation, which resulted in cleaner code architecture and fewer bugs. This approach transformed my programming mindset from just making things work to ensuring they work correctly and can be easily modified later.

What do you see as the advantages and disadvantages of unit testing and test driven development?

The main advantage of unit testing and TDD is the confidence they provide when refactoring or adding new features, as the test suite immediately catches regressions. They also encourage better design by promoting modular, loosely-coupled code. However, the disadvantages include the initial time investment required to write tests and the potential for tests to become maintenance burdens if not well-structured. There's also a risk of over-testing trivial code or creating tests that are too coupled to implementation details.

What strategies would you apply when approaching your own program design and development?

When approaching my own program design and development, I would apply the strategy of starting with clear requirements and breaking them down into small, testable units. I would combine TDD with iterative development, working in small cycles to build and validate each component. Emphasizing separation of concerns through patterns like MVC would be crucial, along with regularly refactoring code based on test feedback and new understanding of the problem domain.

References

Generative AI acknowledgement

Copilot

This file has additional line breaks applied by OnTrack because they contain lines longer than the configured limit. Lines over 1000 characters long have been truncated to limit PDF page count. The orginal submission can be retrieved via the "Download Uploaded Files" function.

```cpp
#include "cheese-shop.h"
#include <fmt/format.h>
using fmt::format;

using std::string;
using std::to_string;

/**
 * Creates a new cheese data structure with the specified parameters
 *
 * @param name The name of the cheese (e.g., "Cheddar", "Gouda")
 * @param weight The weight of cheese in kilograms
 * @param price_per_kg Price per kilogram in cents (to avoid floating point
 ↪  precision issues)
 * @return cheese_data A fully initialized cheese structure
 *
 * @note Uses cents for price to maintain precision in financial calculations
 */
cheese_data new_cheese(string name, double weight, int price_per_kg)
{
    cheese_data cheese;
    cheese.name = name;
    cheese.weight = weight;
    cheese.price_per_kg = price_per_kg;
    return cheese;
}

/**
 * Converts cheese data to a human-readable string representation
 *
 * @param cheese The cheese data to convert to string
 * @param full_details If true, includes weight, price, and total cost;
 *                     if false, returns only the cheese name
 * @return string Formatted string representation of the cheese
 *
 * @example
 *   full_details=true: "Cheddar: 1.50 kg, $15.00 per kg, $22.50 total"
 *   full_details=false: "Cheddar"
 */
string cheese_to_string(const cheese_data &cheese, bool full_details)
{
    if (full_details)
    {
        // Calculate total price and convert to dollars for display
        double total_price = total_cost(cheese) / 100.0;
        return format("{}: {:.2f} kg, ${:.2f} per kg, ${:.2f} total",
                      cheese.name,
                      cheese.weight,
                      cheese.price_per_kg / 100.0,  // Convert cents to dollars
                      total_price);
```

```cpp
    }
    else
    {
        return cheese.name;
    }
}

/**
 * Calculates the total cost of a cheese item based on weight and price per kg
 *
 * @param cheese The cheese data containing weight and price information
 * @return int Total cost in cents (integer to avoid floating point precision issues)
 *
 * @note Returns cost in cents to maintain precision in financial calculations
 */
int total_cost(const cheese_data &cheese)
{
    return static_cast<int>(cheese.weight * cheese.price_per_kg);
}

/**
 * Reduces the weight of a cheese item by the specified amount
 *
 * @param cheese The cheese to reduce weight of (passed by reference for modification)
 * @param amount The amount to reduce in kilograms
 * @return double The actual amount removed (may be less than requested if insufficient stock)
 *
 * @note Handles edge cases: negative amounts return 0, amounts exceeding current weight return remaining weight
 * @example If cheese has 2.0kg and we reduce by 3.0kg, returns 2.0kg and sets weight to 0
 */
double reduce_weight(cheese_data &cheese, double amount)
{
    // Validate input amount
    if (amount <= 0)
        return 0.0;

    // Calculate how much we can actually remove (don't go below zero)
    double actual_removed = amount;
    if (amount > cheese.weight)
    {
        actual_removed = cheese.weight;
    }

    // Update the cheese weight
    cheese.weight -= actual_removed;
    return actual_removed;
}

/**
```

```
 * Increases the weight of a cheese item by the specified amount
 *
 * @param cheese The cheese to increase weight of (passed by reference for
 ↪  modification)
 * @param amount The amount to add in kilograms
 *
 * @note Only positive amounts are processed; negative amounts are ignored
 * @example Used when restocking inventory or receiving supplier deliveries
 */
void increase_weight(cheese_data &cheese, double amount)
{
    if (amount > 0)
    {
        cheese.weight += amount;
    }
}

/**
 * Adds a new cheese to the shop's inventory
 *
 * @param shop The shop data structure to add cheese to
 * @param new_cheese The cheese data to add to the inventory
 *
 * @note Appends the cheese to the end of the cheeses vector
 * @warning Does not check for duplicate cheese names
 */
void add_cheese(shop_data &shop, const cheese_data &new_cheese)
{
    shop.cheeses.push_back(new_cheese);
}

/**
 * Removes a cheese from the shop's inventory by index
 *
 * @param shop The shop data structure to remove cheese from
 * @param index The zero-based index of the cheese to remove
 *
 * @note Performs bounds checking to prevent out-of-range errors
 * @warning Index must be valid (0 <= index < cheeses.size())
 */
void remove_cheese(shop_data &shop, int index)
{
    if (index >= 0 && index < shop.cheeses.size())
    {
        shop.cheeses.erase(shop.cheeses.begin() + index);
    }
}

/**
 * Creates a new customer order with a unique order ID
 *
 * @param customer_name The name of the customer placing the order
 * @return customer_order A new order with auto-incremented ID and pending status
 *
```

```
152  * @note Uses a static counter to generate unique order IDs across the application
153  * @example Order IDs: 1, 2, 3, etc. for sequential orders
154  */
155 customer_order new_customer_order(string customer_name)
156 {
157     static int order_counter = 1;  // Persistent counter for unique order IDs
158 
159     customer_order order;
160     order.order_id = order_counter++;
161     order.customer_name = customer_name;
162     order.fulfilled = false;  // New orders start as unfulfilled
163     return order;
164 }
165 
166 /**
167  * Adds a cheese item to a customer order
168  *
169  * @param order The order to add the cheese to (passed by reference for
     ↪  modification)
170  * @param cheese The cheese data to add to the order
171  *
172  * @note Multiple cheeses of the same type can be added to an order
173  * @warning Cheese price in order may be 0 initially and set during fulfillment
174  */
175 void add_cheese_to_order(customer_order &order, const cheese_data &cheese)
176 {
177     order.cheeses.push_back(cheese);
178 }
179 
180 /**
181  * Removes a cheese item from a customer order by index
182  *
183  * @param order The order to remove the cheese from (passed by reference for
     ↪  modification)
184  * @param index The zero-based index of the cheese to remove from the order
185  *
186  * @note Performs bounds checking to prevent out-of-range errors
187  * @warning Index must be valid (0 <= index < order.cheeses.size())
188  */
189 void remove_cheese_from_order(customer_order &order, int index)
190 {
191     if (index >= 0 && index < order.cheeses.size())
192     {
193         order.cheeses.erase(order.cheeses.begin() + index);
194     }
195 }
196 
197 /**
198  * Calculates the total cost of all cheeses in a customer order
199  *
200  * @param order The customer order containing cheese items
201  * @return int Total cost of the order in cents
202  *
203  * @note Sums the cost of each individual cheese item in the order
```

```
204  * @warning Assumes cheese prices are already set (typically during order
     ↪  fulfillment)
205  */
206 int order_total_cost(const customer_order &order)
207 {
208     int total = 0;
209     for (const auto &cheese : order.cheeses)
210     {
211         total += total_cost(cheese);
212     }
213     return total;
214 }
215 
216 /**
217  * Fulfills a customer order by reducing inventory and marking order as completed
218  *
219  * @param shop The shop data containing current inventory
220  * @param order The order to fulfill (passed by reference for modification)
221  * @return bool True if order was successfully fulfilled, false otherwise
222  *
223  * @note Performs inventory checks before fulfillment to ensure sufficient stock
224  * @note Sets order.fulfilled to true and reduces inventory weights upon success
225  * @warning If any cheese in order is unavailable, entire fulfillment fails
226  */
227 bool fulfill_order(shop_data &shop, customer_order &order)
228 {
229     // Prevent duplicate fulfillment of the same order
230     if (order.fulfilled)
231     {
232         return false;
233     }
234 
235     // Phase 1: Check inventory availability for all cheeses in the order
236     for (const auto &order_cheese : order.cheeses)
237     {
238         bool found = false;
239         // Search shop inventory for matching cheese with sufficient quantity
240         for (const auto &stock_cheese : shop.cheeses)
241         {
242             if (stock_cheese.name == order_cheese.name &&
243                 stock_cheese.weight >= order_cheese.weight)
244             {
245                 found = true;
246                 break;
247             }
248         }
249         // If any cheese is unavailable, cancel the fulfillment
250         if (!found)
251         {
252             return false;
253         }
254     }
255 
256     // Phase 2: Reduce inventory for all cheeses in the order
```

```
257     for (const auto &order_cheese : order.cheeses)
258     {
259         for (auto &stock_cheese : shop.cheeses)
260         {
261             if (stock_cheese.name == order_cheese.name)
262             {
263                 reduce_weight(stock_cheese, order_cheese.weight);
264                 break;
265             }
266         }
267     }
268 
269     // Mark order as fulfilled
270     order.fulfilled = true;
271     return true;
272 }
273 
274 /**
275  * Adds a customer order to the shop's order management system
276  *
277  * @param shop The shop data structure to add the order to
278  * @param order The customer order to add to the shop's records
279  *
280  * @note Appends the order to the end of the orders vector
281  * @warning Does not validate order contents or check for duplicate orders
282  */
283 void add_customer_order(shop_data &shop, const customer_order &order)
284 {
285     shop.orders.push_back(order);
286 }
287 
288 /**
289  * Removes a customer order from the shop's order management system by index
290  *
291  * @param shop The shop data structure to remove the order from
292  * @param index The zero-based index of the order to remove
293  *
294  * @note Performs bounds checking to prevent out-of-range errors
295  * @warning Index must be valid (0 <= index < shop.orders.size())
296  * @warning Removing orders may affect order history and reporting
297  */
298 void remove_customer_order(shop_data &shop, int index)
299 {
300     if (index >= 0 && index < shop.orders.size())
301     {
302         shop.orders.erase(shop.orders.begin() + index);
303     }
304 }
```

This file has additional line breaks applied by OnTrack because they contain lines longer than the configured limit. Lines over 1000 characters long have been truncated to limit PDF page count. The orginal submission can be retrieved via the "Download Uploaded Files" function.

```cpp
#include "catch_amalgamated.hpp"
#include "cheese-shop.h"

using Catch::Approx;

// Customer order tests

TEST_CASE("Customer Order - Create new customer order")
{
    customer_order order = new_customer_order("John Doe");

    REQUIRE(order.order_id > 0);
    REQUIRE(order.customer_name == "John Doe");
    REQUIRE(order.cheeses.size() == 0);
    REQUIRE(order.fulfilled == false);
}

TEST_CASE("Customer Order - Add cheese to order")
{
    customer_order order = new_customer_order("Jane Smith");

    cheese_data cheese1 = new_cheese("Cheddar", 1.5, 2000);
    cheese_data cheese2 = new_cheese("Gouda", 0.5, 1500);

    add_cheese_to_order(order, cheese1);
    add_cheese_to_order(order, cheese2);

    REQUIRE(order.cheeses.size() == 2);
    REQUIRE(order.cheeses[0].name == "Cheddar");
    REQUIRE(order.cheeses[0].weight == Approx(1.5));
    REQUIRE(order.cheeses[1].name == "Gouda");
    REQUIRE(order.cheeses[1].weight == Approx(0.5));
}

TEST_CASE("Customer Order - Remove cheese from order")
{
    customer_order order = new_customer_order("Bob Wilson");

    cheese_data cheese1 = new_cheese("Cheddar", 1.5, 2000);
    cheese_data cheese2 = new_cheese("Gouda", 0.5, 1500);
    cheese_data cheese3 = new_cheese("Brie", 1.0, 3000);

    add_cheese_to_order(order, cheese1);
    add_cheese_to_order(order, cheese2);
    add_cheese_to_order(order, cheese3);

    REQUIRE(order.cheeses.size() == 3);

    remove_cheese_from_order(order, 1); // Remove middle cheese
    REQUIRE(order.cheeses.size() == 2);
```

```
51      REQUIRE(order.cheeses[0].name == "Cheddar");
52      REQUIRE(order.cheeses[1].name == "Brie");
53
54      remove_cheese_from_order(order, 0); // Remove first cheese
55      REQUIRE(order.cheeses.size() == 1);
56      REQUIRE(order.cheeses[0].name == "Brie");
57  }
58
59  TEST_CASE("Customer Order - Calculate total cost")
60  {
61      customer_order order = new_customer_order("Alice Brown");
62
63      cheese_data cheese1 = new_cheese("Cheddar", 1.5, 2000); // 3000 cents
64      cheese_data cheese2 = new_cheese("Gouda", 0.5, 1500);   // 750 cents
65      cheese_data cheese3 = new_cheese("Brie", 1.0, 3000);    // 3000 cents
66
67      add_cheese_to_order(order, cheese1);
68      add_cheese_to_order(order, cheese2);
69      add_cheese_to_order(order, cheese3);
70
71      int total = order_total_cost(order);
72      REQUIRE(total == 6750); // 3000 + 750 + 3000
73  }
74
75  TEST_CASE("Customer Order - Fulfill order with sufficient stock")
76  {
77      shop_data shop;
78
79      // Add stock
80      add_cheese(shop, new_cheese("Cheddar", 2.0, 2000));
81      add_cheese(shop, new_cheese("Gouda", 1.0, 1500));
82
83      // Create order
84      customer_order order = new_customer_order("Customer");
85      add_cheese_to_order(order, new_cheese("Cheddar", 1.5, 2000));
86      add_cheese_to_order(order, new_cheese("Gouda", 0.5, 1500));
87
88      REQUIRE(fulfill_order(shop, order) == true);
89      REQUIRE(order.fulfilled == true);
90
91      // Check stock was reduced
92      REQUIRE(shop.cheeses[0].weight == Approx(0.5)); // Cheddar: 2.0 - 1.5 = 0.5
93      REQUIRE(shop.cheeses[1].weight == Approx(0.5)); // Gouda: 1.0 - 0.5 = 0.5
94  }
95
96  TEST_CASE("Customer Order - Cannot fulfill order with insufficient stock")
97  {
98      shop_data shop;
99
100     // Add limited stock
101     add_cheese(shop, new_cheese("Cheddar", 1.0, 2000)); // Only 1.0 kg available
102
103     // Create order requiring more than available
104     customer_order order = new_customer_order("Customer");
```

```
105     add_cheese_to_order(order, new_cheese("Cheddar", 1.5, 2000)); // Need 1.5 kg
106
107     REQUIRE(fulfill_order(shop, order) == false);
108     REQUIRE(order.fulfilled == false);
109
110     // Check stock was not reduced
111     REQUIRE(shop.cheeses[0].weight == Approx(1.0));
112 }
113
114 TEST_CASE("Customer Order - Cannot fulfill already fulfilled order")
115 {
116     shop_data shop;
117     add_cheese(shop, new_cheese("Cheddar", 2.0, 2000));
118
119     customer_order order = new_customer_order("Customer");
120     add_cheese_to_order(order, new_cheese("Cheddar", 1.0, 2000));
121
122     // First fulfillment should succeed
123     REQUIRE(fulfill_order(shop, order) == true);
124
125     // Second fulfillment should fail
126     REQUIRE(fulfill_order(shop, order) == false);
127 }
128
129 TEST_CASE("Shop - Manage customer orders")
130 {
131     shop_data shop;
132
133     customer_order order1 = new_customer_order("Customer 1");
134     customer_order order2 = new_customer_order("Customer 2");
135
136     add_customer_order(shop, order1);
137     add_customer_order(shop, order2);
138
139     REQUIRE(shop.orders.size() == 2);
140     REQUIRE(shop.orders[0].customer_name == "Customer 1");
141     REQUIRE(shop.orders[1].customer_name == "Customer 2");
142
143     remove_customer_order(shop, 0);
144     REQUIRE(shop.orders.size() == 1);
145     REQUIRE(shop.orders[0].customer_name == "Customer 2");
146 }
```

This file has additional line breaks applied by OnTrack because they contain lines longer than the configured limit. Lines over 1000 characters long have been truncated to limit PDF page count. The orginal submission can be retrieved via the "Download Uploaded Files" function.

```
#ifndef CHEESE_SHOP_H
#define CHEESE_SHOP_H

#include <string>
#include <vector>

using std::string;
using std::vector;

/**
 * Data about a cheese within an order or in stock.
 *
 * @field name The name of the cheese.
 * @field weight The weight of the cheese in stock (kg).
 * @field price_per_kg The price of the cheese per kg (cents).
 */
struct cheese_data
{
    string name;
    double weight;
    int price_per_kg;
};

/**
 * Customer order containing multiple cheese items.
 *
 * @field order_id Unique identifier for the order
 * @field customer_name Name of the customer
 * @field cheeses List of cheeses in the order
 * @field fulfilled Whether the order has been fulfilled
 */
struct customer_order
{
    int order_id;
    string customer_name;
    vector<cheese_data> cheeses;
    bool fulfilled;
};

/**
 * Data about the cheese shop - the stock on hand and customer orders.
 *
 * @field cheeses The list of cheeses in stock.
 * @field orders The list of customer orders.
 * @field next_order_id The next available order ID.
 */
struct shop_data
{
    vector<cheese_data> cheeses;
    vector<customer_order> orders;
```

```
51      int next_order_id;
52  };
53
54  // Existing cheese functions
55  cheese_data new_cheese(string name = "", double weight = 0.0, int price_per_kg = 0);
56  string cheese_to_string(const cheese_data &cheese, bool full_details = false);
57  int total_cost(const cheese_data &cheese);
58  double reduce_weight(cheese_data &cheese, double amount);
59  void increase_weight(cheese_data &cheese, double amount);
60
61  // Shop management functions
62  void add_cheese(shop_data &shop, const cheese_data &new_cheese);
63  void remove_cheese(shop_data &shop, int index);
64
65  // Customer order functions
66  /**
67   * Create a new customer order.
68   *
69   * @param customer_name The name of the customer.
70   * @return customer_order The new order with a unique ID.
71   */
72  customer_order new_customer_order(string customer_name = "");
73
74  /**
75   * Add a cheese to a customer order.
76   *
77   * @param order The order to add cheese to.
78   * @param cheese The cheese to add.
79   */
80  void add_cheese_to_order(customer_order &order, const cheese_data &cheese);
81
82  /**
83   * Remove a cheese from a customer order.
84   *
85   * @param order The order to remove cheese from.
86   * @param index The index of the cheese to remove.
87   */
88  void remove_cheese_from_order(customer_order &order, int index);
89
90  /**
91   * Calculate the total cost of a customer order.
92   *
93   * @param order The order to calculate cost for.
94   * @return int The total cost in cents.
95   */
96  int order_total_cost(const customer_order &order);
97
98  /**
99   * Fulfill a customer order, reducing stock accordingly.
100  *
101  * @param shop The shop data containing stock.
102  * @param order The order to fulfill.
103  * @return bool True if order was successfully fulfilled, false otherwise.
104  */
```

```
105 bool fulfill_order(shop_data &shop, customer_order &order);
106
107 /**
108  * Add a customer order to the shop.
109  *
110  * @param shop The shop data.
111  * @param order The order to add.
112  */
113 void add_customer_order(shop_data &shop, const customer_order &order);
114
115 /**
116  * Remove a customer order from the shop.
117  *
118  * @param shop The shop data.
119  * @param index The index of the order to remove.
120  */
121 void remove_customer_order(shop_data &shop, int index);
122
123 #endif
```

17 More Python

At this stage you should have a solid understanding of programming, and the skills needed to work on programming projects. As you progress, you really need to start learning a few languages well. You have a reasonable start with the syntax underlying C-style languages (like C#, C/C++, and Java) and you have explored a little of Python. As Python is a very popular language, and is used in a number of units that follow FIT1045, this task asks you to explore the language further and build your confidence with it. The great thing is, Python is very easy to learn, particularly if you already have a good understanding of the programming fundamentals.

Date	Author	Comment
2025/08/06 20:27	Ziyue Meng	Planned date adjusted to 10 Oct.
2025/10/12 16:34	Ziyue Meng	Working On It
2025/10/13 01:38	Ziyue Meng	Ready for Feedback
2025/10/21 16:10	Ganesh Krishnasamy	Complete

Monash

Introduction to Programming

OnTrack Submission

More Python

Submitted By:
Ziyue Meng
zmen0034
2025/10/13 01:38

Tutor:
Ganesh Krishnasamy

October 13, 2025

More Python

Name: Ziyue Meng
Student ID: 36035432
Date: 12/10/2025

Learning Journey and Evidence

Prior Knowledge and Experience

I have a solid foundation in several key aspects of programming. I've mastered the use of pointers and references. Pointers are variables that store memory addresses, which allow for direct memory manipulation and efficient data passing between functions. References, on the other hand, act as an alias for an existing variable, providing a more intuitive way to access and modify data in some cases.

I'm also well - versed in loops. For example, the for loop is useful when you know exactly how many times you want to iterate. It has a concise syntax where you can initialize a counter, set a condition for continuation, and update the counter in each iteration. The while loop is more flexible, as it continues as long as a given condition is true, making it suitable for situations where the number of iterations is not fixed in advance. The do - while loop is similar to the while loop, but it executes the loop body at least once before checking the condition.

Enumerations (enum) are a user - defined data type in programming. They allow you to create a set of named constants. For instance, if you are creating a program about the days of the week, you can define an enum like enum Days {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY}; which makes the code more readable and maintainable.

Regarding structures, I can define a structure to encapsulate related data. For example, a "Student" struct containing name, ID, and score. I'm able to properly access its members, which is crucial for handling complex data entities in programming.

When it comes to arrays, I have a good grasp of their usage. I know how to declare and initialize both one - dimensional and two - dimensional arrays. For a one - dimensional array, it can be done like int numbers[5] = {1, 2, 3, 4, 5}; and for a two - dimensional array, something like int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}} works. I'm familiar with iterating through array elements using loops, which is essential for processing the data they hold. Arrays are great for storing collections of similar data, such as a list of student scores or a series of temperatures. I also understand the importance of being cautious with array index bounds to avoid accessing elements outside the valid range, which can lead to unexpected program behavior.

Study to Acquire, Refresh, and/or Extend Capability

To study and master this topic I did the following:

7.1.1. Formatted String Literals

Formatted string literals (also called f-strings for short) let you include the value of Python expressions inside a string by prefixing the string with `f` or `F` and writing expressions as `{expression}`.

An optional format specifier can follow the expression. This allows greater control over how the value is formatted. The following example rounds pi to three places after the decimal:

```
>>> import math
>>> print(f'The value of pi is approximately {math.pi:.3f}.')
The value of pi is approximately 3.142.
```

Passing an integer after the `':'` will cause that field to be a minimum number of characters wide. This is useful for making columns line up.

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}
>>> for name, phone in table.items():
...     print(f'{name:10} ==> {phone:10d}')
...
Sjoerd     ==>       4127
Jack       ==>       4098
Dcab       ==>       7678
```

Other modifiers can be used to convert the value before it is formatted. `'!a'` applies `ascii()`, `'!s'` applies `str()`, and `'!r'` applies `repr()`:

```
>>> animals = 'eels'
>>> print(f'My hovercraft is full of {animals}.')
My hovercraft is full of eels.
>>> print(f'My hovercraft is full of {animals!r}.')
My hovercraft is full of 'eels'.
```

The `=` specifier can be used to expand an expression to the text of the expression, an equal sign, then the representation of the evaluated expression:

```
>>> bugs = 'roaches'
>>> count = 13
>>> area = 'living room'
>>> print(f'Debugging {bugs=} {count=} {area=}')
Debugging bugs='roaches' count=13 area='living room'
```

test.py > ...

```python
# Trying basic f-string usage
name = "John"
age = 25
print(f"My name is {name}, I'm {age} years old")

# Trying number formatting
import math
pi = math.pi
print(f"Pi to 2 decimal places: {pi:.2f}")
print(f"Pi to 4 decimal places: {pi:.4f}")

# Trying field width and alignment
scores = {"Alice": 95, "Bob": 87, "Charlie": 92}
for student, score in scores.items():
    print(f"{student:8} ==> {score:3} points")
```

```
ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/learnforp12
$ python test.py
My name is John, I'm 25 years old
Pi to 2 decimal places: 3.14
Pi to 4 decimal places: 3.1416
Alice    ==>  95 points
Bob      ==>  87 points
Charlie  ==>  92 points
```

Advantages of f-strings

1. Concise and intuitive syntax, just add `f` before the string
2. Support writing expressions directly in curly braces, like `{math.pi:.3f}`
3. Convenient for debugging, using `{variable=}` directly displays variable name and value
4. Generally better performance than format() method

```python
# Positional arguments usage
print("{0} likes {1}, and {1} also likes {0}".format("John", "Mary"))

# Keyword arguments usage
print("This {book} costs {price} dollars".format(book="Python Programming", price=59.8))

# Mixed usage
info = {"name": "Lisa", "age": 20, "city": "New York"}
print("Student info: {name}, {age} years old, from {city}".format(**info))
```

```
ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/learnforp12
$ python test.py
John likes Mary, and Mary also likes John
This Python Programming costs 59.8 dollars
Student info: Lisa, 20 years old, from New York
```

Flexibility of format() Method

- Supports both positional and keyword arguments, allowing finer format control
- Can format through dictionary unpacking `**dict`
- Suitable for complex formatting needs, especially when reusing the same values

```python
# Using rjust, ljust, center
text = "Python"
print("Right aligned:" + text.rjust(10, "*"))
print("Left aligned:" + text.ljust(10, "*"))
print("Center aligned:" + text.center(10, "*"))

# Creating a simple table
headers = ["Name", "Age", "Score"]
data = [("John", 18, 95), ("Mary", 17, 88), ("Tom", 19, 92)]

print("|".join(header.center(8) for header in headers))
print("-" * 30)
for name, age, score in data:
    print(f"{name.center(8)}|{str(age).center(8)}|{str(score).center(8)}")
```

```
ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/learnforp12
$ python test.py
Right aligned:****Python
Left aligned:Python****
Center aligned:**Python**
  Name   |   Age    | Score
------------------------------
  John   |    18    |   95
  Mary   |    17    |   88
  Tom    |    19    |   92
```

Appropriate Use Cases for Manual Formatting

- `str.rjust()`, `str.ljust()`, `str.center()` are suitable for simple alignment needs
- `str.zfill()` is particularly useful for zero-padding numbers
- Practical when complete control over output format is needed

```python
def display_student_records():
    """Display student grade records"""
    # Simulating data read from a file
    students = [
        {"name": "John", "math": 85, "english": 92, "science": 78},
        {"name": "Lisa", "math": 92, "english": 88, "science": 95},
        {"name": "Mike", "math": 78, "english": 85, "science": 90}
    ]

    # Using f-string and format together to create a table
    header = f"{'Name':^8} | {'Math':^6} | {'English':^6} | {'Science':^6} | {'Average':^8}"
    print(header)
    print("-" * 50)

    for student in students:
        avg_score = (student["math"] + student["english"] + student["science"]) / 3
        # Using format method to ensure number alignment
        row = "{name:^8} | {math:^6} | {english:^6} | {science:^6} | {avg:^8.2f}".format(
            name=student["name"],
            math=student["math"],
            english=student["english"],
            science=student["science"],
            avg=avg_score
        )
        print(row)

# Run the example
display_student_records()
```

```
ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/learnforp12
$ python test.py
  Name   |  Math  | English | Science | Average
--------------------------------------------------
  John   |   85   |   92    |   78    |  85.00
  Lisa   |   92   |   88    |   95    |  91.67
  Mike   |   78   |   85    |   90    |  84.33
```

Practical Application Techniques

- Calculating column widths is important when creating tables
- Mixing different formatting methods can achieve better results
- Number formatting (like `.2f`) makes output more professional

Unit Test

```python
# string_formatter.py
def calculate_average_score(scores_dict):
    """Calculate average score from a dictionary of scores"""
    math_score = scores_dict.get("math", 0)
    english_score = scores_dict.get("english", 0)
    science_score = scores_dict.get("science", 0)
    return (math_score + english_score + science_score) / 3

def format_student_info(name, age, average_score):
    """Format student information using f-string"""
    return f"Student: {name}, Age: {age}, Average: {average_score:.2f}"
```

The first function, `calculate_average_score`, takes a dictionary of student scores as input. It retrieves the values for three subjects - math, English, and science - using the dictionary's `get` method, which returns a default value of 0 if any subject key is missing. The function then calculates and returns the arithmetic mean of these three scores by adding them together and dividing by 3.

The second function, `format_student_info`, accepts three parameters: a student's name, age, and average score. It uses an f-string to format this information into a readable string. The f-string incorporates the name and age directly, while the average score is formatted to display with exactly two decimal places using the `:.2f` format specifier. This ensures consistent numerical formatting in the output string, which follows the pattern "Student: [name], Age: [age], Average: [score]".

```python
import unittest
from string_formatter import calculate_average_score, format_student_info

class TestStudentFormatter(unittest.TestCase):

    def test_calculate_average_score(self):
        """Test average score calculation"""
        test_scores = {"math": 85, "english": 92, "science": 78}
        result = calculate_average_score(test_scores)
        expected = (85 + 92 + 78) / 3
        self.assertAlmostEqual(result, expected, places=2)

    def test_calculate_average_with_missing_scores(self):
        """Test average score calculation with missing subjects"""
        test_scores = {"math": 85, "english": 92}  # Missing science
        result = calculate_average_score(test_scores)
        expected = (85 + 92 + 0) / 3
        self.assertAlmostEqual(result, expected, places=2)

    def test_format_student_info(self):
        """Test student information formatting"""
        result = format_student_info("John", 20, 85.6666)
        expected = "Student: John, Age: 20, Average: 85.67"
        self.assertEqual(result, expected)

    def test_format_student_info_rounding(self):
        """Test formatting rounding"""
        result = format_student_info("Alice", 19, 90.555)
        self.assertIn("90.56", result)  # Should round to two decimal places

if __name__ == "__main__":
    unittest.main()
```

```
ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/learnforp12
$ python test.py
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
```

I'm learning data structures in Python.

5.1. More on Lists

The list data type has some more methods. Here are all of the methods of list objects:

`list.`**`append`**`(`*`x`*`)`

Add an item to the end of the list. Similar to `a[len(a):] = [x]`.

`list.`**`extend`**`(`*`iterable`*`)`

Extend the list by appending all the items from the iterable. Similar to `a[len(a):] = iterable`.

`list.`**`insert`**`(`*`i`*`, `*`x`*`)`

Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

`list.`**`remove`**`(`*`x`*`)`

Remove the first item from the list whose value is equal to *x*. It raises a `ValueError` if there is no such item.

`list.`**`pop`**`([`*`i`*`])`

Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. It raises an `IndexError` if the list is empty or the index is outside the list range.

`list.`**`clear`**`()`

Remove all items from the list. Similar to `del a[:]`.

`list.`**`index`**`(`*`x`*`[, `*`start`*`[, `*`end`*`]])`

Return zero-based index of the first occurrence of *x* in the list. Raises a `ValueError` if there is no such item.

The optional arguments *start* and *end* are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the *start* argument.

`list.clear()`
Remove all items from the list. Similar to `del a[:]`.

`list.index(x[, start[, end]])`
Return zero-based index of the first occurrence of *x* in the list. Raises a `ValueError` if there is no such item.

The optional arguments *start* and *end* are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the *start* argument.

`list.count(x)`
Return the number of times *x* appears in the list.

`list.sort(*, key=None, reverse=False)`
Sort the items of the list in place (the arguments can be used for sort customization, see `sorted()` for their explanation).

`list.reverse()`
Reverse the elements of the list in place.

`list.copy()`
Return a shallow copy of the list. Similar to `a[:]`.

```python
"""
This script demonstrates my learning process of Python list methods
through practical examples and experimentation.
"""

def demonstrate_list_methods():
    """Demonstrate various list methods through practical examples"""

    # Starting with a basic list
    fruits = ['apple', 'banana', 'orange']
    print(f"Original list: {fruits}")

    # 1. append() - Add single item to the end
    fruits.append('grape')
    print(f"After append('grape'): {fruits}")

    # 2. extend() - Add multiple items from another iterable
    more_fruits = ['mango', 'pineapple']
    fruits.extend(more_fruits)
    print(f"After extend(['mango', 'pineapple']): {fruits}")

    # 3. insert() - Insert at specific position
    fruits.insert(1, 'kiwi')  # Insert at index 1
    print(f"After insert(1, 'kiwi'): {fruits}")

    # 4. remove() - Remove first occurrence of an item
    fruits.remove('banana')
    print(f"After remove('banana'): {fruits}")

    # 5. pop() - Remove and return item at specific position
    removed_fruit = fruits.pop(2)
    print(f"After pop(2): {fruits}, removed: '{removed_fruit}'")

    # 6. index() - Find position of an item
    orange_index = fruits.index('orange')
    print(f"Index of 'orange': {orange_index}")
```

```python
    # 7. count() - Count occurrences of an item
    fruits.append('apple')  # Add duplicate for demonstration
    apple_count = fruits.count('apple')
    print(f"Count of 'apple': {apple_count}")
    print(f"Current list: {fruits}")

    # 8. sort() - Sort the list in place
    fruits.sort()
    print(f"After sort(): {fruits}")

    # 9. reverse() - Reverse the list in place
    fruits.reverse()
    print(f"After reverse(): {fruits}")

    # 10. copy() - Create a shallow copy
    fruits_copy = fruits.copy()
    fruits_copy.append('watermelon')
    print(f"Original: {fruits}")
    print(f"Copy with 'watermelon': {fruits_copy}")

    # 11. clear() - Remove all items
    fruits_copy.clear()
    print(f"After clear(): {fruits_copy}")
```

```python
def practical_list_application():
    """Show a practical application using multiple list methods"""

    # Simulating a student grade management system
    grades = [85, 92, 78, 90, 88]
    print(f"\n--- Student Grade Management ---")
    print(f"Original grades: {grades}")

    # Add a new grade
    grades.append(95)
    print(f"After adding new grade: {grades}")

    # Add multiple makeup grades
    makeup_grades = [82, 87]
    grades.extend(makeup_grades)
    print(f"After makeup exams: {grades}")

    # Sort grades
    grades.sort()
    print(f"Sorted grades: {grades}")

    # Find highest and lowest
    highest = grades[-1]  # Last item after sort
    lowest = grades[0]    # First item after sort
    print(f"Highest grade: {highest}")
    print(f"Lowest grade: {lowest}")

    # Calculate average
    average = sum(grades) / len(grades)
    print(f"Average grade: {average:.2f}")

    # Count grades above average
    above_average = len([grade for grade in grades if grade > average])
    print(f"Grades above average: {above_average}")
```

```python
if __name__ == "__main__":
    print("=== Learning Python List Methods ===\n")

    # Demonstrate individual methods
    demonstrate_list_methods()

    # Show practical application
    practical_list_application()

    print("\n=== Learning Complete ===")
```

Through learning Python's various list methods, I have gained a comprehensive understanding of how to efficiently manipulate and manage list data. I've mastered how to use `append()` and `extend()` for adding elements, `insert()` for placing elements at specific positions, and `remove()` and `pop()` for element removal. The `sort()` and `reverse()` methods enable me to rearrange list order, while `index()` and `count()` help me search and analyze list contents. Most importantly, I've understood the practical application scenarios for these methods, such as how to use them in a student grade management system to add, sort, and analyze data. These list methods significantly enhance my ability and efficiency in handling collection data within Python, providing me with powerful tools for data manipulation in real-world programming tasks.

Here is a small example using a dictionary:

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'jack': 4098, 'sape': 4139, 'guido': 4127}
>>> tel['jack']
4098
>>> tel['irv']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'irv'
>>> print(tel.get('irv'))
None
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'jack': 4098, 'guido': 4127, 'irv': 4127}
>>> list(tel)
['jack', 'guido', 'irv']
>>> sorted(tel)
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```

The dict() constructor builds dictionaries directly from sequences of key-value pairs:

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

In addition, dict comprehensions can be used to create dictionaries from arbitrary key and value expressions:

```
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

When the keys are simple strings, it is sometimes easier to specify pairs using keyword arguments:

```
>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

```python
"""
This script demonstrates my learning process of Python dictionaries
and various looping techniques through practical examples.
"""

def demonstrate_dictionary_operations():
    """Show various dictionary operations and methods"""

    # Creating dictionaries using different methods
    print("=== Dictionary Creation Methods ===")

    # Method 1: Using curly braces
    student_scores = {'Alice': 85, 'Bob': 92, 'Charlie': 78}
    print(f"1. Using curly braces: {student_scores}")

    # Method 2: Using dict() constructor with key-value pairs
    student_scores2 = dict([('David', 88), ('Eva', 95), ('Frank', 81)])
    print(f"2. Using dict() with pairs: {student_scores2}")

    # Method 3: Using keyword arguments
    student_scores3 = dict(Grace=90, Henry=87, Iris=93)
    print(f"3. Using keyword arguments: {student_scores3}")

    # Method 4: Dictionary comprehension
    squares = {x: x**2 for x in range(1, 6)}
    print(f"4. Dictionary comprehension: {squares}")
```

```python
def demonstrate_dictionary_methods():
    """Demonstrate common dictionary methods"""

    print("\n=== Dictionary Operations ===")
    inventory = {'apples': 10, 'bananas': 15, 'oranges': 8}
    print(f"Original inventory: {inventory}")

    # Adding and updating items
    inventory['grapes'] = 12  # Add new item
    inventory['apples'] = 20  # Update existing item
    print(f"After updates: {inventory}")

    # Accessing values
    print(f"Number of bananas: {inventory['bananas']}")
    print(f"Number of mangoes (using get): {inventory.get('mangoes', 0)}")

    # Checking existence
    print(f"'oranges' in inventory: {'oranges' in inventory}")
    print(f"'pears' not in inventory: {'pears' not in inventory}")

    # Removing items
    removed_value = inventory.pop('bananas')
    print(f"Removed bananas: {removed_value}")
    print(f"After pop: {inventory}")

    # Getting keys and values
    print(f"All items: {list(inventory.items())}")
    print(f"All keys: {list(inventory.keys())}")
    print(f"All values: {list(inventory.values())}")
```

```python
def demonstrate_looping_techniques():
    """Show various looping techniques with dictionaries and sequences"""

    print("\n=== Looping Techniques ===")

    # 1. Looping through dictionary items
    knights = {'gallahad': 'the pure', 'robin': 'the brave', 'lancelot': 'the knight'}
    print("1. Dictionary looping with items():")
    for name, title in knights.items():
        print(f"   {name}: {title}")

    # 2. Using enumerate() for index and value
    fruits = ['apple', 'banana', 'cherry']
    print("\n2. Sequence looping with enumerate():")
    for index, fruit in enumerate(fruits):
        print(f"   Index {index}: {fruit}")

    # 3. Using zip() to loop multiple sequences
    names = ['Alice', 'Bob', 'Charlie']
    scores = [85, 92, 78]
    subjects = ['Math', 'Science', 'English']
    print("\n3. Multiple sequences with zip():")
    for name, score, subject in zip(names, scores, subjects):
        print(f"   {name} scored {score} in {subject}")

    # 4. Looping in reverse
    print("\n4. Looping in reverse with reversed():")
    for i in reversed(range(1, 6)):
        print(f"   Countdown: {i}")
```

```python
    # 5. Looping in sorted order
    grades = ['B', 'A', 'C', 'A', 'B', 'D']
    print("\n5. Looping in sorted order:")
    for grade in sorted(grades):
        print(f"   Grade: {grade}")

    # 6. Looping unique elements in sorted order
    print("\n6. Unique elements in sorted order:")
    for unique_grade in sorted(set(grades)):
        print(f"   Unique grade: {unique_grade}")
```

```python
def practical_dictionary_application():
    """Show a practical application using dictionaries and looping"""

    print("\n=== Practical Application: Student Management System ===")

    # Student database using dictionary
    students = {
        'Alice': {'age': 20, 'major': 'Computer Science', 'gpa': 3.8},
        'Bob': {'age': 21, 'major': 'Mathematics', 'gpa': 3.5},
        'Charlie': {'age': 19, 'major': 'Physics', 'gpa': 3.9},
        'Diana': {'age': 22, 'major': 'Computer Science', 'gpa': 3.7}
    }

    # Display all students
    print("All Students:")
    for name, info in students.items():
        print(f"  {name}: {info['age']} years old, {info['major']}, GPA: {info['gpa']}")

    # Find Computer Science students
    print("\nComputer Science Students:")
    for name, info in students.items():
        if info['major'] == 'Computer Science':
            print(f"  {name} (GPA: {info['gpa']})")

    # Calculate average GPA by major
    print("\nAverage GPA by Major:")
    majors_gpa = {}
    for name, info in students.items():
        major = info['major']
        if major not in majors_gpa:
            majors_gpa[major] = []
        majors_gpa[major].append(info['gpa'])
```

```python
    for major, gpas in majors_gpa.items():
        avg_gpa = sum(gpas) / len(gpas)
        print(f"  {major}: {avg_gpa:.2f}")

def demonstrate_data_filtering():
    """Show data filtering techniques"""

    print("\n=== Data Filtering Example ===")

    # Raw data with some invalid entries
    raw_temperatures = [23.5, 25.1, float('NaN'), 22.8, float('NaN'), 24.9, 21.7]

    # Filter out NaN values
    valid_temperatures = []
    for temp in raw_temperatures:
        if not (isinstance(temp, float) and math.isnan(temp)):
            valid_temperatures.append(temp)

    print(f"Raw data: {raw_temperatures}")
    print(f"Filtered data: {valid_temperatures}")
    print(f"Average temperature: {sum(valid_temperatures)/len(valid_temperatures):.1f}°C")

import math
```

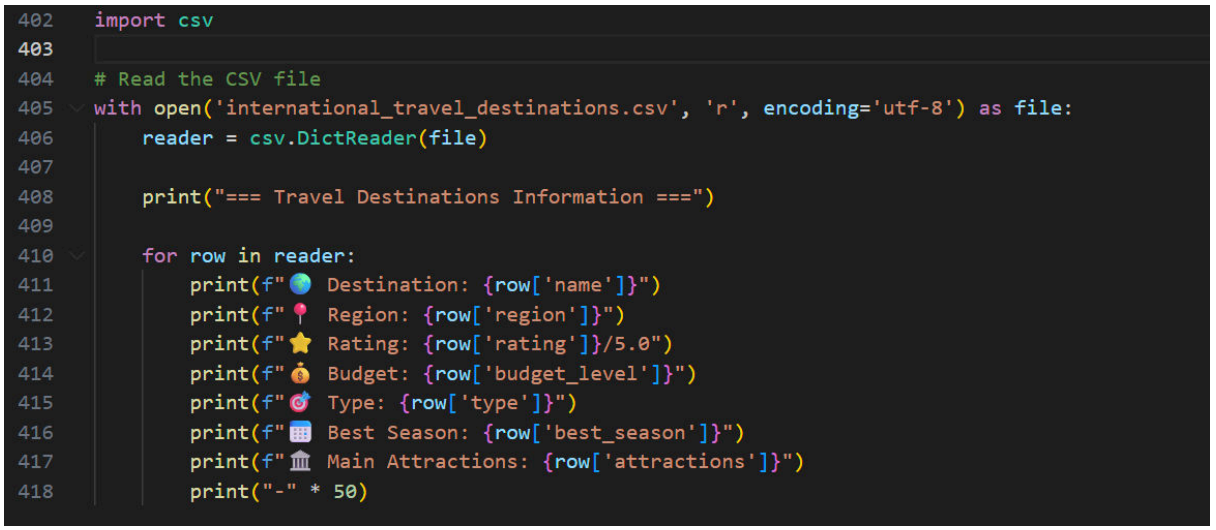
```python
if __name__ == "__main__":
    print("=== Learning Python Dictionaries and Looping Techniques ===\n")

    demonstrate_dictionary_operations()
    demonstrate_dictionary_methods()
    demonstrate_looping_techniques()
    practical_dictionary_application()
    demonstrate_data_filtering()

    print("\n=== Learning Complete ===")
```

```
ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/learnforp12
$ python test.py
=== Learning Python Dictionaries and Looping Techniques ===

=== Dictionary Creation Methods ===
1. Using curly braces: {'Alice': 85, 'Bob': 92, 'Charlie': 78}
2. Using dict() with pairs: {'David': 88, 'Eva': 95, 'Frank': 81}
3. Using keyword arguments: {'Grace': 90, 'Henry': 87, 'Iris': 93}
4. Dictionary comprehension: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

=== Dictionary Operations ===
Original inventory: {'apples': 10, 'bananas': 15, 'oranges': 8}
After updates: {'apples': 20, 'bananas': 15, 'oranges': 8, 'grapes': 12}
Number of bananas: 15
Number of mangoes (using get): 0
'oranges' in inventory: True
'pears' not in inventory: True
Removed bananas: 15
After pop: {'apples': 20, 'oranges': 8, 'grapes': 12}
All items: [('apples', 20), ('oranges', 8), ('grapes', 12)]
All keys: ['apples', 'oranges', 'grapes']
All values: [20, 8, 12]

=== Looping Techniques ===
1. Dictionary looping with items():
   gallahad: the pure
   robin: the brave
   lancelot: the knight

2. Sequence looping with enumerate():
   Index 0: apple
   Index 1: banana
   Index 2: cherry

3. Multiple sequences with zip():
   Alice scored 85 in Math
   Bob scored 92 in Science
   Charlie scored 78 in English

4. Looping in reverse with reversed():
   Countdown: 5
   Countdown: 4
   Countdown: 3
   Countdown: 2
   Countdown: 1
```

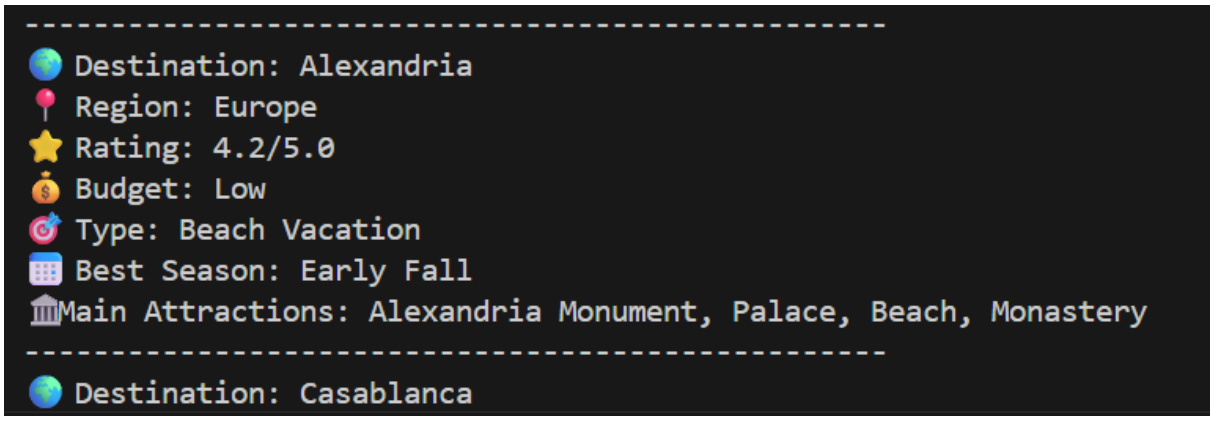
```
5. Looping in sorted order:
   Grade: A
   Grade: A
   Grade: B
   Grade: B
   Grade: C
   Grade: D

6. Unique elements in sorted order:
   Unique grade: A
   Unique grade: B
   Unique grade: C
   Unique grade: D

=== Practical Application: Student Management System ===
All Students:
  Alice: 20 years old, Computer Science, GPA: 3.8
  Bob: 21 years old, Mathematics, GPA: 3.5
  Charlie: 19 years old, Physics, GPA: 3.9
  Diana: 22 years old, Computer Science, GPA: 3.7

Computer Science Students:
  Alice (GPA: 3.8)
  Diana (GPA: 3.7)

Average GPA by Major:
  Computer Science: 3.75
  Mathematics: 3.50
  Physics: 3.90

=== Data Filtering Example ===
Raw data: [23.5, 25.1, nan, 22.8, nan, 24.9, 21.7]
Filtered data: [23.5, 25.1, 22.8, 24.9, 21.7]
Average temperature: 23.6°C

=== Learning Complete ===

ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/learnforp12
$
```

Through this learning process, I've gained a comprehensive understanding of Python dictionaries and advanced looping techniques. I learned that dictionaries are key-value pair collections that allow efficient data retrieval using immutable keys. I mastered various dictionary creation methods, operations like adding, updating, and removing items, and useful methods like `get()`, `items()`, `keys()`, and `values()`. For looping, I discovered powerful techniques including `enumerate()` for index-value pairs, `zip()` for iterating multiple sequences simultaneously, `reversed()` for backward iteration, and `sorted()` for ordered processing. I also learned how to combine `set()` with `sorted()` to iterate through unique elements in order. These skills enable me to handle complex data structures more effectively and write more efficient, readable Python code for real-world applications like student management systems and data filtering tasks.

Learn about CSV:



```python
import csv

# Read the CSV file
with open('international_travel_destinations.csv', 'r', encoding='utf-8') as file:
    reader = csv.DictReader(file)

    print("=== Travel Destinations Information ===")

    for row in reader:
        print(f"🌍 Destination: {row['name']}")
        print(f"📍 Region: {row['region']}")
        print(f"⭐ Rating: {row['rating']}/5.0")
        print(f"💰 Budget: {row['budget_level']}")
        print(f"🎯 Type: {row['type']}")
        print(f"🗓️ Best Season: {row['best_season']}")
        print(f"🏛️ Main Attractions: {row['attractions']}")
        print("-" * 50)
```

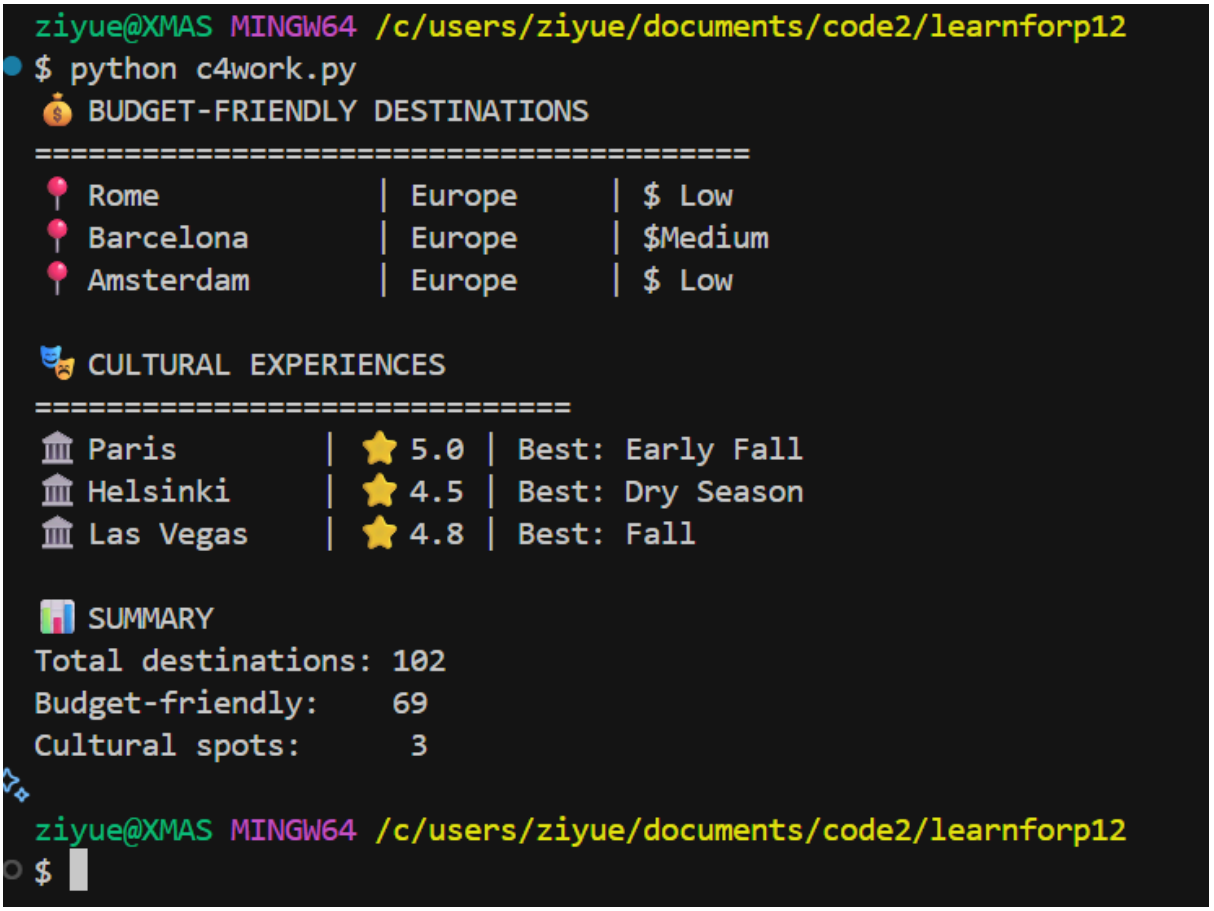


Through this practical exercise with CSV file handling in Python, I gained valuable experience in reading and processing structured data efficiently. I learned how to use the csv module's DictReader to access data by column names, which makes the code more readable and maintainable. The process of selectively extracting specific information like destination details, ratings, and attractions taught me how to filter and present relevant data from larger datasets. This hands-on practice reinforced my understanding of file I/O operations and demonstrated how Python can transform raw data into organized, meaningful information. These skills are fundamental for working with real-world data in various applications, from travel databases to business analytics.

```python
"""
A simple travel planner program demonstrating:
- Lists usage
- String formatting
- CSV file reading
"""

import csv

def main():
    # I learned how to read data from CSV files using csv.DictReader
    # This automatically creates dictionaries for each row with column headers as keys
    destinations = []
    with open('international_travel_destinations.csv', 'r', encoding='utf-8') as file:
        reader = csv.DictReader(file)
        for row in reader:
            destinations.append(row)

    # Use lists to store and process data
    budget_destinations = []
    cultural_destinations = []

    for dest in destinations:
        # Filter by budget level
        if dest['budget_level'].lower() in ['low', 'medium']:
            budget_destinations.append(dest)

        # Filter by type
        if 'cultural' in dest['type'].lower():
            cultural_destinations.append(dest)

    # I learned to use f-strings with formatting for clean, aligned output
    # Using field widths {:15} and alignment {:^6} makes data presentation professional
    print("💰 BUDGET-FRIENDLY DESTINATIONS")
    print("=" * 40)
    for dest in budget_destinations[:3]:  # Show top 3
        print(f"📍 {dest['name']:15} | {dest['region']:10} | ${dest['budget_level']:^6}")

    print(f"\n🎭 CULTURAL EXPERIENCES")
    print("=" * 30)
```

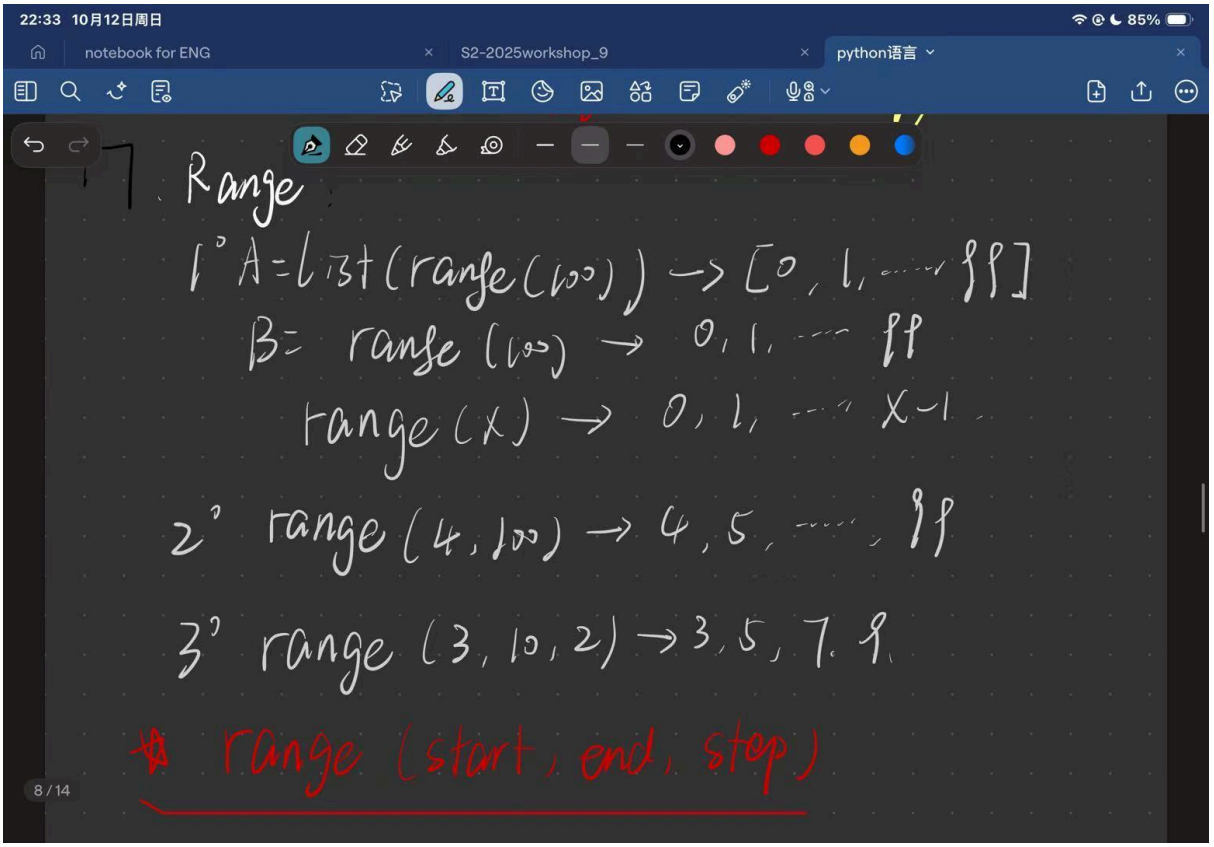
```python
    for dest in budget_destinations[:3]:  # Show top 3
        print(f"📍 {dest['name']:15} | {dest['region']:10} | ${dest['budget_level']:^6}")

    print(f"\n🎭 CULTURAL EXPERIENCES")
    print("=" * 30)
    for dest in cultural_destinations[:3]:
        rating = float(dest['rating'])
        print(f"🏛️  {dest['name']:12} | ⭐ {rating:.1f} | Best: {dest['best_season']:8}")

    # Putting everything together - using lists for counting, formatting for display
    total = len(destinations)
    budget_count = len(budget_destinations)
    cultural_count = len(cultural_destinations)

    print(f"\n📊 SUMMARY")
    print(f"Total destinations: {total:2d}")
    print(f"Budget-friendly:    {budget_count:2d}")
    print(f"Cultural spots:     {cultural_count:2d}")

if __name__ == "__main__":
    main()
```

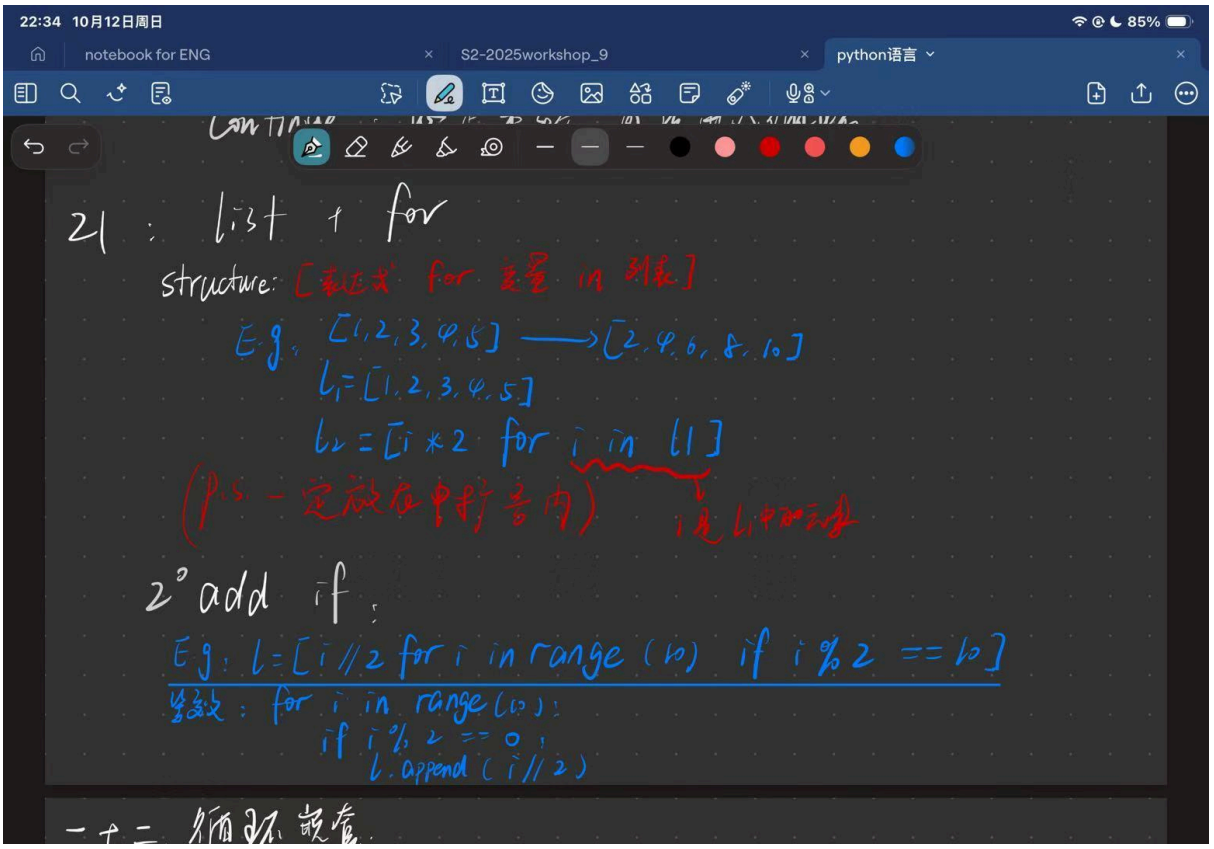
OUTPUT:


ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/learnforp12
$ python c4work.py
BUDGET-FRIENDLY DESTINATIONS
Rome | Europe | $ Low
Barcelona | Europe | $Medium
Amsterdam | Europe | $ Low
CULTURAL EXPERIENCES
Paris | 5.0 | Best: Early Fall
Helsinki | 4.5 | Best: Dry Season
Las Vegas | 4.8 | Best: Fall
SUMMARY
Total destinations: 102
Budget-friendly: 69
Cultural spots: 3
ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/learnforp12
$

My Python Notes

Add your own notes on the Python language here - this can then be a good resource for you as you go forward into future units that use Python.





Reflection

What is the most important thing you learned from this and why?

The most important thing I learned was how to effectively combine different Python concepts - file handling, data structures, and string formatting - to solve practical problems. This integration is crucial because in real-world programming, these skills are rarely used in isolation; understanding how they work together enables me to build more functional and efficient applications.

What strategies did you apply when extending your Python capability?

I employed a progressive learning strategy: starting with basic syntax and simple examples, then gradually building complexity by combining concepts. I practiced with hands-on coding exercises, referred to official documentation for deeper understanding, and consistently tested my code to identify and fix errors, which reinforced my learning through practical application.

How did your understanding of the programming concepts help you with this task?

My foundational knowledge of programming concepts like data structures, control flow, and functions made learning Python-specific features much easier. Understanding general programming principles allowed me to focus on Python's unique syntax and libraries rather than struggling with basic computational thinking, which significantly accelerated my learning process.

What will you do differently/the same when you have to learn your next programming language?

I will maintain the same approach of hands-on practice and project-based learning, but I'll place more emphasis on understanding the language's specific idioms and best practices from the beginning. I'll also dedicate more time to learning the standard library ecosystem, as I now recognize how crucial ready-made solutions are for efficient programming in any language.

References

Generative AI acknowledgement

Copilot is used to find errors in my code.

This file has additional line breaks applied by OnTrack because they contain lines longer than the configured limit. Lines over 1000 characters long have been truncated to limit PDF page count. The orginal submission can be retrieved via the "Download Uploaded Files" function.

```
"""
A simple travel planner program demonstrating:
- Lists usage
- String formatting
- CSV file reading
"""

import csv

def main():
    # I learned how to read data from CSV files using csv.DictReader
    # This automatically creates dictionaries for each row with column headers as keys
    destinations = []
    with open('international_travel_destinations.csv', 'r', encoding='utf-8') as file:
        reader = csv.DictReader(file)
        for row in reader:
            destinations.append(row)

    # Use lists to store and process data
    budget_destinations = []
    cultural_destinations = []

    for dest in destinations:
        # Filter by budget level
        if dest['budget_level'].lower() in ['low', 'medium']:
            budget_destinations.append(dest)

        # Filter by type
        if 'cultural' in dest['type'].lower():
            cultural_destinations.append(dest)

    # I learned to use f-strings with formatting for clean, aligned output
    # Using field widths {:15} and alignment {:^6} makes data presentation professional
    print(" BUDGET-FRIENDLY DESTINATIONS")
    print("=" * 40)
    for dest in budget_destinations[:3]:  # Show top 3
        print(f" {dest['name']:15} | {dest['region']:10} | ${dest['budget_level']:^6}")

    print(f"\n CULTURAL EXPERIENCES")
    print("=" * 30)
    for dest in cultural_destinations[:3]:
        rating = float(dest['rating'])
        print(f"  {dest['name']:12} |  {rating:.1f} | Best: {dest['best_season']:8}")

    # Putting everything together - using lists for counting, formatting for display
```

```
    total = len(destinations)
    budget_count = len(budget_destinations)
    cultural_count = len(cultural_destinations)

    print(f"\n SUMMARY")
    print(f"Total destinations: {total:2d}")
    print(f"Budget-friendly:    {budget_count:2d}")
    print(f"Cultural spots:     {cultural_count:2d}")

if __name__ == "__main__":
    main()
```

18 Active Engagement

Learning is done best with others. This task aims to encourage and reward you for your active engagement during the semester. Show us how you have engaged with other students and the teaching team, online and in person.

Date	Author	Comment
2025/08/06 20:28	Ziyue Meng	Planned date adjusted to 17 Oct.
2025/10/16 14:54	Ziyue Meng	Working On It
2025/10/16 17:51	Ziyue Meng	Ready for Feedback
2025/10/21 16:02	Ganesh Krishnasamy	Discuss

Monash

Introduction to Programming

OnTrack Submission

Active Engagement

Submitted By:
Ziyue Meng
zmen0034
2025/10/16 17:51

Tutor:
Ganesh Krishnasamy

October 16, 2025

Active Engagement

Name: Ziyue Meng
Student ID: 36035432
Date: 16/10/2025

Learning Journey and Evidence

Prior Knowledge and Experience

I have a solid foundation in several key aspects of programming. I've mastered the use of pointers and references. Pointers are variables that store memory addresses, which allow for direct memory manipulation and efficient data passing between functions. References, on the other hand, act as an alias for an existing variable, providing a more intuitive way to access and modify data in some cases.

I'm also well - versed in loops. For example, the for loop is useful when you know exactly how many times you want to iterate. It has a concise syntax where you can initialize a counter, set a condition for continuation, and update the counter in each iteration. The while loop is more flexible, as it continues as long as a given condition is true, making it suitable for situations where the number of iterations is not fixed in advance. The do - while loop is similar to the while loop, but it executes the loop body at least once before checking the condition.

Enumerations (enum) are a user - defined data type in programming. They allow you to create a set of named constants. For instance, if you are creating a program about the days of the week, you can define an enum like enum Days {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY}; which makes the code more readable and maintainable.

Regarding structures, I can define a structure to encapsulate related data. For example, a "Student" struct containing name, ID, and score. I'm able to properly access its members, which is crucial for handling complex data entities in programming.

When it comes to arrays, I have a good grasp of their usage. I know how to declare and initialize both one - dimensional and two - dimensional arrays. For a one - dimensional array, it can be done like int numbers[5] = {1, 2, 3, 4, 5}; and for a two - dimensional array, something like int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}} works. I'm familiar with iterating through array elements using loops, which is essential for processing the data they hold. Arrays are great for storing collections of similar data, such as a list of student scores or a series of temperatures. I also understand the importance of being cautious with array index bounds to avoid accessing elements outside the valid range, which can lead to unexpected program behavior.

Key Interactions

Some key interactions from my learning journey:

Ziyue Meng 15/09/2025

Ready for Feedback

Ganesh Krishnasamy 22/09/2025

GK Can also include what you learned so far in the previous task (P1 - C1) before attempting this task in the prior knowledge and experience.

Fix and Resubmit

Explanation with the issue with this code for finding the max value in an array is incorrect.

GK Did not explain what is the issue with the code for the sorting problem.

Ziyue Meng 25/09/2025

Ready for Feedback

Ganesh Krishnasamy 28/09/2025

```
switch (option)
{
    case 1:
        add_product(my_store, products);
        break;
    case 2:
        remove_product(my_store, products);
        break;
    case 3:
        update_products(my_store, products);
        break;
    case 4:
        print_store_summary(my_store, products);
        break;
    case 5:
        printf("Exiting program.\n");
        break;
    default:
        printf("Invalid choice. Please try again.\n");
        break;
}
```

Each product will have:

- a name
- a cost price
- a sale price
- a number on hand (how many of these products the store has)

your struct product_data is incorrect.

The shop will have:

- a number of products (the system needs to support up to 100 products)
- total sales (dollars)
- total profit (dollars)

your struct for store_data is incorrect.

Ziyue Meng 29/09/2025

Ready for Feedback

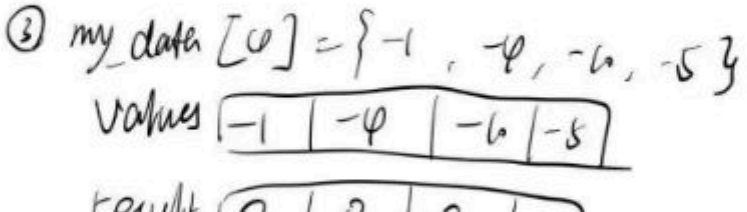
Ziyue Meng 01/10/2025

I have modified my work and reupload it, thank you for your guidance!

Ganesh Krishnasamy 01/10/2025

Your hand solution for bubble sort is incorrect. Please re-submit.

Fix and Resubmit



GK did not answer this question yet.

Fix and Resubmit

Ziyue Meng 02/10/2025

This is the answer

This program defines a double_array struct and uses bubble sort to arrange its elements in ascending order. It repeatedly compares adjacent values, swapping them if out of order. Each outer loop pass moves the largest remaining value to the end, reducing the range until the array is fully sorted. Just change the upper limit of j to size-1, otherwise it will involve redundant data.

Ready for Feedback

I have modified my work and reupload it, thank you for your guidance!

Ganesh Krishnasamy 04/10/2025

Fix and Resubmit

There are no comments in your

Fix and Resubmit

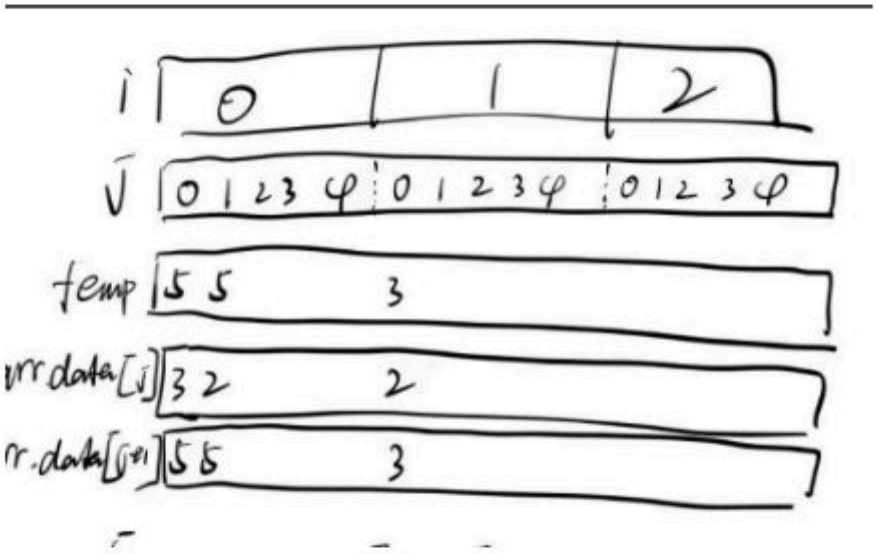
There are no comments in your code. Please revise and add documentation to show your intent and help others read it.

```
case 1:
    store.products[index].name = read_string("Enter new name: ");
    break;
case 2:
    store.products[index].costPrice = read_double("Enter new cost price: ");
    break;
case 3:
    store.products[index].sellingPrice = read_double("Enter new selling
    ↪ price: ");
    break;
case 4:
    store.products[index].quantity = read_integer("Enter new quantity: ");
    break;
case 5:
    printf("Quitting update.\n");
    break;
default:
    printf("Invalid option.\n");
    break;
```

GK

You should use enum that to map to these switch cases in your test your knowledge program.

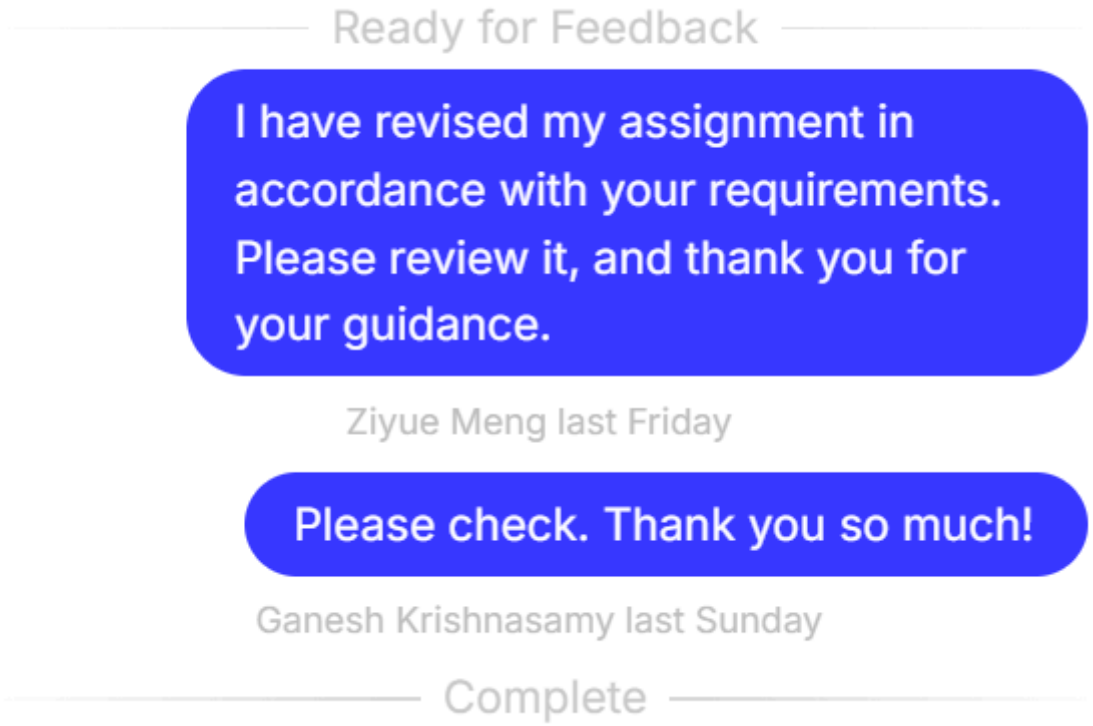
Fix and Resubmit



still incorrect.

GK

pls re-visit and submit it



Throughout my learning journey in this course, I have been truly fortunate to receive consistent and meticulous guidance from my instructor, Ganesh. His attention to detail was remarkable—he carefully reviewed every submission, from code structure and algorithm logic to documentation and naming conventions. Each piece of feedback was specific and constructive, whether it was pointing out an incorrect struct definition, suggesting the use of enums for better readability, or reminding me to add comments for clarity. His patience and dedication helped me not only correct my mistakes but also understand the underlying principles. Thanks to his supportive and thoughtful mentoring, I was able to steadily improve and eventually succeed in completing the tasks with confidence.

Hi Ziyue.

There are no comments in your code. Please revise and add documentation to show your intent and help others read it.

GK

I can't see any engagement with with the guided tour and concepts for Structring Data. At least, engagement with one of the guided tour and capture the learning journey.

Fix and Resubmit

Ziyue Meng 06/09/2025

Ready for Feedback

I have modified my work and reupload it, thank you for your guidance!

Ganesh Krishnasamy 09/09/2025

GK

You have included many screenshots of your learning, but it would be better if you can describe what you have learned from doing all these that shown in the screenshots.

Fix and Resubmit

Ziyue Meng 09/09/2025

My learning journey was greatly enriched by the detailed feedback from my instructor. His reminders to add code comments and engage with guided tours pushed me to improve both my technical documentation and conceptual understanding. Through this process, I learned to not only fix errors but also articulate my learning progress more effectively.

During the project development, I actively engaged in code discussions with my classmates to refine our understanding and solutions. We regularly reviewed each other's code segments, debated different implementation approaches, and troubleshooted issues together. These collaborative sessions not only helped me identify gaps in my own logic but also exposed me to alternative programming perspectives and techniques. Through explaining my code to peers and receiving their constructive feedback, I strengthened both my technical communication skills and problem-solving abilities, ultimately leading to more robust and well-structured code for our project.

Reflection

What is the most important thing you learned from this and why?

The most important thing I learned is the value of active engagement and collaborative discussion in deepening my understanding of the course material. During classes, I made an effort not only to listen but also to ask questions and participate in group discussions. Outside of class, I regularly reviewed and explored concepts with peers. Through these interactions, I realized that complex topics—such as data structures or algorithm logic—became much clearer when explained and discussed with others, far more than when studying alone.

How has your engagement helped you develop?

My active involvement helped me strengthen my own understanding while also improving my ability to articulate and explain technical concepts. When helping classmates clarify their doubts, I had to reorganize my own knowledge and identify gaps in my understanding. For example, during workshops, when I explained how to solve a particular problem, I was pushed to fully grasp the concept before I could make it understandable to others—which in turn reinforced my own learning.

What was the most impactful moment or series of events for your learning and development?

The most impactful experience was during the pre-exam review sessions, when the professor walked us through the practice test. He didn't just go over the answers—he guided us through the underlying principles and common pitfalls. Those sessions helped connect scattered concepts into a coherent whole. Thanks to that review, I was able to clarify previously confusing topics and felt much more confident facing similar questions in the actual exam.

Any particular takeaways from this experience overall?

This experience reinforced that learning is not just about *receiving* information—it's also about *output* and *interaction*. Actively participating in class, daring to ask questions, engaging in discussions with peers, and valuing instructors' guidance all contributed significantly to my academic growth. I plan to continue learning in this collaborative and proactive way in the future.

References

Generative AI acknowledgement

19 Custom Program - Progress

In D2, you have outlined a plan and had this signed off as acceptable. Through this task, you need to discuss your projects progress and demonstrate how you are working on it with the teaching team.

Date	Author	Comment
2025/08/06 20:25	Ziyue Meng	Planned date adjusted to 26 Sep.
2025/10/15 15:20	Ziyue Meng	Ready for Feedback
2025/10/21 16:01	Ganesh Krishnasamy	Thanks for demonstrating your progress your D project. Please fix the errors/bugs that you are facing. Please think on how you are planning to extent your work to HD level.
2025/10/21 16:01	Ganesh Krishnasamy	Complete

Monash

Introduction to Programming

OnTrack Submission

Custom Program - Progress

Submitted By:
Ziyue Meng
zmen0034
2025/10/15 15:20

Tutor:
Ganesh Krishnasamy

October 15, 2025

Custom Project

Name: ZIyue Meng
Student ID: 36035432
Date: 5/10/2025

Project Pitch (D2)

Programming Language(s): Python
Using GenAI: Copilot
Target Grade:HD

Overview:

This project develops an intelligent travel planning assistant that provides personalized destination recommendations using machine learning algorithms. The system processes a comprehensive dataset of 100 international travel destinations collected from online sources, offering users a wide selection of global travel options.

The application features a user-friendly graphical interface with three main components: a destination browser for exploring travel locations, a recommendation system that suggests destinations based on user preferences, and basic data visualization for understanding recommendation results. Users can set their travel preferences including destination type, budget level, and preferred regions to receive customized suggestions.

The core functionality is built using Python's scikit-learn library, implementing K-Nearest Neighbors algorithm for finding similar destinations and Random Forest regression for predicting user ratings. The system handles data management through pandas and presents results through an intuitive Tkinter-based interface.

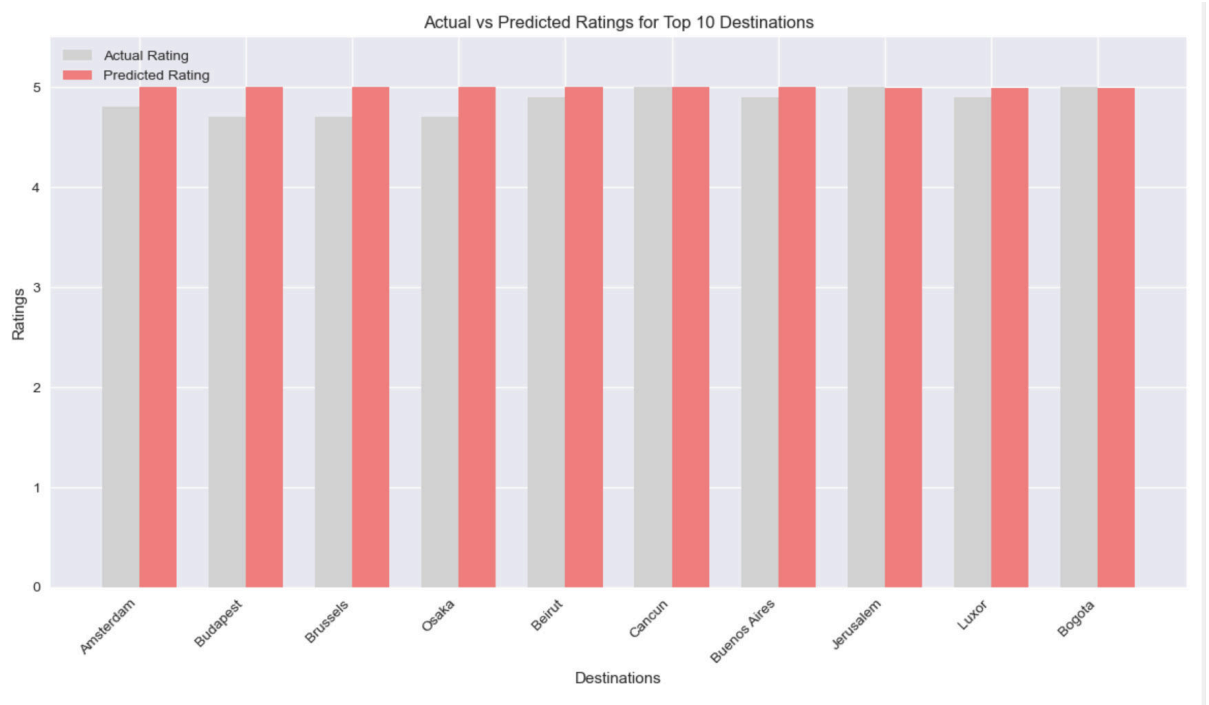
visualization.py > ...

```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from typing import List, Dict
import numpy as np

class Visualization:
    def __init__(self):
        plt.style.use('seaborn-v0_8')
        self.fig_size = (12, 8)

    def plot_predicted_ratings(self, df: pd.DataFrame):
        """Plot predicted ratings"""
        if 'predicted_rating' not in df.columns or df.empty:
            return None

        top_15 = df.head(15)

        fig, ax = plt.subplots(figsize=(12, 8))
        y_pos = np.arange(len(top_15))

        bars = ax.barh(y_pos, top_15['predicted_rating'], color='lightblue', alpha=0.7)
        ax.set_yticks(y_pos)
        ax.set_yticklabels(top_15['name'])
        ax.set_xlabel('Predicted Rating')
        ax.set_title('Top 15 Recommended Destinations - Predicted Ratings')
        ax.set_xlim(0, 5.5)

        # Add value labels
        for i, (bar, rating) in enumerate(zip(bars, top_15['predicted_rating'])):
            ax.text(bar.get_width() + 0.1, bar.get_y() + bar.get_height()/2,
                    f'{rating:.2f}', va='center', fontweight='bold')

        plt.tight_layout()
        return fig

    def plot_ratings_comparison(self, df: pd.DataFrame):
        """Plot comparison between actual and predicted ratings"""
        if 'predicted_rating' not in df.columns or df.empty:
            return None
```

OUTPUT PORTS bash

ents/project_english

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.13.2



main.py > ...

```python
import tkinter as tk
from tkinter import ttk, messagebox, scrolledtext
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from travel_system import TravelPlanningSystem

class TravelPlannerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Intelligent Travel Planning Assistant")
        self.root.geometry("1200x800")

        self.system = TravelPlanningSystem()

        # Create main frame
        self.main_frame = ttk.Frame(root)
        self.main_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Create notebook
        self.notebook = ttk.Notebook(self.main_frame)
        self.notebook.pack(fill=tk.BOTH, expand=True)

        self.create_destinations_tab()
        self.create_recommendations_tab()
        self.create_visualization_tab()

        # Load initial data
        self.refresh_destinations()
```

HD Extension Ideas:

- Automated Data Integration
  - Implement web scraping to dynamically collect and update destination data from travel websites
  - Develop real-time data updating system with automated data validation
  - Integrate multiple online data sources for comprehensive destination information
- Advanced Machine Learning Features
  - Implement collaborative filtering using matrix factorization for improved recommendations
  - Develop content-based filtering that analyzes destination features and user preferences
  - Create hybrid recommendation system combining multiple ML approaches
  - Implement model performance evaluation and comparison framework
- Enhanced Visualization System
  - Develop interactive data dashboards with multiple chart types
  - Implement geospatial visualization showing destinations on maps
  - Create comparative analysis charts for destination features and ratings
  - Build real-time visualization updates based on user interactions

- Intelligent Recommendation Features
    - Implement recommendation explanation system showing why destinations are suggested
    - Develop multi-criteria recommendation considering season, duration, and activities
    - Create similarity analysis showing how destinations relate to each other
    - Build preference learning system that adapts based on user feedback
- Advanced User Features
    - Implement user profile management with preference history
    - Develop saved recommendations and travel planning functionality
    - Create destination comparison tools with side-by-side feature analysis
    - Build export functionality for recommendations and travel plans
- System Optimization
    - Implement caching system for faster recommendation generation
    - Develop automated data preprocessing and cleaning pipelines
    - Create performance monitoring and optimization features
    - Build modular architecture for easy feature expansion
-

How did you come up with your project pitch?

This project was inspired by the challenge of planning travel in an increasingly complex world with countless destination options. Traditional travel planning often involves sifting through overwhelming amounts of information without personalized guidance. I identified an opportunity to apply machine learning techniques to create a smart recommendation system that understands individual preferences.

The concept combines practical travel needs with data science applications, addressing the real problem of destination choice overload. By focusing on personalized recommendations, the system helps users discover destinations they might otherwise overlook. The use of a comprehensive dataset from online sources ensures the recommendations are based on extensive, real-world travel information.

The HD extensions were designed to push the project beyond basic functionality into a sophisticated travel intelligence platform. These enhancements focus on improving recommendation accuracy, providing deeper insights through advanced visualization, and creating a more engaging user experience through interactive features. The project demonstrates how machine learning can transform everyday tasks like travel planning into intelligent, data-driven experiences.

Project Progress (D3)

Prior Knowledge and Experience

I have a solid foundation in several key aspects of programming. I've mastered the use of pointers and references. Pointers are variables that store memory addresses, which allow for direct memory manipulation and efficient data passing between functions. References, on the other hand, act as an alias for an existing variable, providing a more intuitive way to access and modify data in some cases.

I'm also well - versed in loops. For example, the for loop is useful when you know exactly how many times you want to iterate. It has a concise syntax where you can initialize a counter, set a condition for continuation, and update the counter in each iteration. The while loop is more flexible, as it continues as long as a given condition is true, making it suitable for situations where the number of iterations is not fixed in advance. The do - while loop is similar to the while loop, but it executes the loop body at least once before checking the condition.

Enumerations (enum) are a user - defined data type in programming. They allow you to create a set of named constants. For instance, if you are creating a program about the days of the week, you can define an enum like enum Days {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY}; which makes the code more readable and maintainable.

Regarding structures, I can define a structure to encapsulate related data. For example, a "Student" struct containing name, ID, and score. I'm able to properly access its members, which is crucial for handling complex data entities in programming.

When it comes to arrays, I have a good grasp of their usage. I know how to declare and initialize both one - dimensional and two - dimensional arrays. For a one - dimensional array, it can be done like int numbers[5] = {1, 2, 3, 4, 5}; and for a two - dimensional array, something like int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}} works. I'm familiar with iterating through array elements using loops, which is essential for processing the data they hold. Arrays are great for storing collections of similar data, such as a list of student scores or a series of temperatures. I also understand the importance of being cautious with array index bounds to avoid accessing elements outside the valid range, which can lead to unexpected program behavior.
4

Iterations and Study to Create Project and Extend Capabilities

To build this project, and extend my programming capabilities, I did the following:

The thought process begins with an analysis of the problem: traditional travel planning tools often lack personalized recommendations, and users can feel overwhelmed when faced with too many choices. To address this, the proposed solution involves building an intelligent recommendation system that utilizes machine

learning algorithms and provides visual analytics to help users make more informed and personalized travel decisions.

Core Module Breakdown

1. data_manager.py

   Data Layer: Handles all data operations

2. travel_system.py

   Business Layer: Processes core logic

3. ml_recommender.py

   Algorithm Layer: Machine learning recommendations

4. visualization.py

   Presentation Layer: Data visualization

5. main.py

   Interface Layer: User interaction

```python
class DataManager:
    def __init__(self):
        self.destinations_file = "destinations.json"
        self.users_file = "users.json"
        self.csv_file = "international_travel_destinations.csv"
        self.init_data()

    def init_data(self):
        """Initialize data from CSV file"""
        if os.path.exists(self.csv_file):
            self.load_from_csv()
        else:
            self.init_default_data()
```

The code initializes a `DataManager` class designed to handle multiple data sources, including JSON files for destinations and users, as well as a CSV file for international travel destinations. It implements a flexible data loading strategy by first checking for the existence of the CSV file. If available, it prioritizes loading data from this CSV; if not, it falls back to initializing default data, ensuring system adaptability and robustness.

```python
    def load_from_csv(self):
        """Load destinations from CSV file"""
        try:
            df = pd.read_csv(self.csv_file)
            destinations = []

            for _, row in df.iterrows():
                destination = {
                    "id": int(row['id']),
                    "name": row['name'],
                    "type": row['type'],
                    "attractions": row['attractions'].split(', ') if pd.notna(row['attractions']) else [],
                    "best_season": row['best_season'],
                    "specialty": row['specialties'].split(', ') if pd.notna(row['specialties']) else [],
                    "budget_level": row['budget_level'],
                    "climate": row['climate'],
                    "rating": float(row['rating']),
                    "tags": row['tags'].split(', ') if pd.notna(row['tags']) else [],
                    "region": row.get('region', ''),
                    "ideal_duration": row.get('ideal_duration', ''),
                    "language": row.get('language', ''),
                    "currency": row.get('currency', ''),
                    "safety_level": row.get('safety_level', ''),
                    "family_friendly": bool(row.get('family_friendly', True)),
                    "adventure_level": row.get('adventure_level', ''),
                    "nightlife_rating": row.get('nightlife_rating', ''),
                    "shopping_rating": row.get('shopping_rating', '')
                }
                destinations.append(destination)
```

```python
                self.save_destinations(destinations)
                print(f"Loaded {len(destinations)} destinations from CSV")

            except Exception as e:
                print(f"Error loading CSV: {e}")
                self.init_default_data()
```

The `load_from_csv` function reads and processes data from a CSV file, converting it into the system's required format. It utilizes pandas to load the CSV data into a DataFrame, then iterates through each row to perform data cleaning and format transformation. During this process, it extracts and converts various fields—such as splitting the 'attractions' string into a list—and compiles the processed data into a list of destination dictionaries. If successful, it saves the converted data; if any errors occur during loading or processing, it catches the exception, outputs an error message, and falls back to initializing the system with default data.

```python
class TravelPlanningSystem:
    def __init__(self):
        self.data_manager = DataManager()
        self.ml_recommender = MLRecommender(self.data_manager)
        self.visualization = Visualization()
        self.current_user = "default_user"

    def add_destination(self, destination_data: Dict) -> bool:
        """Add a destination"""
        destinations = self.data_manager.load_destinations()

        # Generate new ID
        new_id = max([d['id'] for d in destinations], default=0) + 1
        destination_data['id'] = new_id

        destinations.append(destination_data)
        self.data_manager.save_destinations(destinations)

        # Retrain models
        self.ml_recommender.train_knn_model()
        return True

    def get_all_destinations(self) -> List[Dict]:
        """Get all destinations"""
        return self.data_manager.load_destinations()
```

The code establishes a `TravelPlanningSystem` class that functions as the central coordinator, integrating all key modules into a complete workflow. During initialization, it creates instances of the `DataManager`, `MLRecommender`, and `Visualization` classes, and sets a default user. Crucially, it includes a bridge method, `get_personalized_recommendations`, which connects the data management and machine learning components. This method retrieves the user's preferences and then leverages the ML recommender to generate and return predicted ratings, demonstrating the system's interconnected architecture.

```python
    def update_user_preferences(self, preferences: Dict):
        """Update user preferences"""
        users_data = self.data_manager.load_users()
        users_data['user_preferences'][self.current_user] = preferences
        self.data_manager.save_users(users_data)

        # Retrain rating prediction model
        self.ml_recommender.train_rating_model(preferences)

    def get_user_preferences(self) -> Dict:
        """Get user preferences"""
        users_data = self.data_manager.load_users()
        return users_data['user_preferences'].get(self.current_user, {})
```

The code implements a user-centric function, `update_user_preferences`, which manages a core input for the recommendation system. When a user's preferences are updated, the system first loads all user data, then specifically updates the current user's preference record before saving the modified data. Crucially, the system is designed to immediately retrain the machine learning model using these new preferences upon any update, ensuring that the recommendations remain dynamically aligned with the user's latest interests.

```python
def prepare_features(self, df: pd.DataFrame) -> pd.DataFrame:
    """Prepare features for machine learning"""
    df_encoded = df.copy()

    # Encode categorical variables
    categorical_columns = ['type', 'best_season', 'budget_level', 'climate']
    for col in categorical_columns:
        if col in df_encoded.columns:
            self.label_encoders[col] = LabelEncoder()
            df_encoded[col + '_encoded'] = self.label_encoders[col].fit_transform(df_encoded[col].fillna('Unknown'))

    # Create tag features (convert tags list to features)
    all_tags = set()
    for tags in df_encoded['tags']:
        if isinstance(tags, list):
            all_tags.update(tags)

    for tag in all_tags:
        df_encoded[f'tag_{tag}'] = df_encoded['tags'].apply(
            lambda x: 1 if isinstance(x, list) and tag in x else 0
        )

    # Select numerical features
    feature_columns = [col for col in df_encoded.columns if col.endswith('_encoded') or col.startswith('tag_')]
    feature_columns.extend(['rating'])

    return df_encoded[feature_columns].fillna(0)
```

The `prepare_features` function performs essential feature engineering to transform raw data into a format suitable for machine learning. It begins by creating a copy of the input DataFrame, then addresses the challenge of converting categorical text data into numerical values. Using Label Encoders, it processes categorical columns like 'type' and 'climate', creating new encoded columns for each.

A key innovation in this process is the transformation of the 'tags' list into a feature matrix. The function first identifies all unique tags across the dataset, then creates binary indicator columns for each tag, marking their presence (1) or absence (0) for

each destination. The final output is a cleaned DataFrame containing only the engineered numerical features, with any missing values filled with zeros.

```
    def train_knn_model(self, n_neighbors=5):
        """Train KNN recommendation model"""
        df = self.data_manager.get_destinations_dataframe()
        if len(df) == 0:
            return

        features = self.prepare_features(df)
        self.knn_model = NearestNeighbors(n_neighbors=min(n_neighbors, len(df)), metric='cosine')
        self.knn_model.fit(features)

    def train_kmeans_model(self, n_clusters=5):
        """Train KMeans clustering model"""
        df = self.data_manager.get_destinations_dataframe()
        if len(df) == 0:
            return

        features = self.prepare_features(df)
        n_clusters = min(n_clusters, len(df))
        self.kmeans_model = KMeans(n_clusters=n_clusters, random_state=42)
        self.kmeans_model.fit(features)
        return self.kmeans_model.labels_

    def train_rating_model(self, user_preferences: Dict):
        """Train rating prediction model"""
        df = self.data_manager.get_destinations_dataframe()
        if len(df) == 0 or not user_preferences:
            return
```

This code defines three distinct machine learning training methods within the recommendation system:

The `train_knn_model` method builds a K-Nearest Neighbors model using cosine similarity to identify similar destinations, automatically adjusting the number of neighbors based on available data size.

The `train_kmeans_model` method performs destination clustering using K-Means algorithm, grouping similar travel spots into clusters for pattern discovery and recommendation diversification.

The `train_rating_model` method prepares a rating prediction model that leverages user preferences to forecast potential ratings for various destinations.

All three methods incorporate safety checks to ensure sufficient data exists before model training, and utilize the feature engineering pipeline to convert raw destination data into machine-readable numerical features.

```python
def train_rating_model(self, user_preferences: Dict):
    """Train rating prediction model"""
    df = self.data_manager.get_destinations_dataframe()
    if len(df) == 0 or not user_preferences:
        return

    # Create simulated ratings based on user preferences
    features = self.prepare_features(df)

    # Simple rule-based rating prediction
    y = df['rating'].copy()

    # Adjust ratings based on user preferences
    for idx, destination in df.iterrows():
        preference_boost = 0
        if user_preferences.get('preferred_type') and user_preferences['preferred_type'] in destination['type']:
            preference_boost += 0.5
        if user_preferences.get('budget_level') and user_preferences['budget_level'] == destination['budget_level']:
            preference_boost += 0.3
        if user_preferences.get('preferred_region') and user_preferences['preferred_region'] in str(destination.get('region', '')):
            preference_boost += 0.2

        y.iloc[idx] = min(5.0, y.iloc[idx] + preference_boost)

    self.rating_model = RandomForestRegressor(n_estimators=20, random_state=42)
    self.rating_model.fit(features, y)
```

The `train_rating_model` function implements a hybrid approach to train a rating prediction model that combines content-based filtering with machine learning. It begins by retrieving destination data and applying feature engineering. The core innovation lies in creating simulated ratings based on user preferences - it starts with existing destination ratings, then applies rule-based adjustments where ratings are boosted when destinations match the user's preferred type, budget level, or region. These preference-based boosts are carefully weighted and capped at the maximum rating of 5.0. Finally, the enhanced ratings are used to train a Random Forest Regressor, creating a personalized recommendation model that learns from both inherent destination features and explicit user preferences.

```python
class TravelPlannerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Intelligent Travel Planning Assistant")
        self.root.geometry("1200x800")

        self.system = TravelPlanningSystem()

        # Create main frame
        self.main_frame = ttk.Frame(root)
        self.main_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Create notebook
        self.notebook = ttk.Notebook(self.main_frame)
        self.notebook.pack(fill=tk.BOTH, expand=True)

        self.create_destinations_tab()
        self.create_recommendations_tab()
        self.create_visualization_tab()

        # Load initial data
        self.refresh_destinations()

```

The code initializes the main application class `TravelPlannerApp`, which creates the graphical user interface for the Intelligent Travel Planning Assistant. The interface is built using Tkinter, with a window size of 1200x800 pixels. The class instantiates the core `TravelPlanningSystem` to handle backend operations and organizes the interface using a notebook widget with multiple tabs for different functionalities - destinations, recommendations, and visualizations. Finally, it loads initial data by calling the refresh method to populate the destinations tab when the application launches.

```python
def create_recommendations_tab(self):
    """Create recommendations tab"""
    frame = ttk.Frame(self.notebook)
    self.notebook.add(frame, text="Recommendations")

    # User preferences
    pref_frame = ttk.LabelFrame(frame, text="User Preferences")
    pref_frame.pack(fill=tk.X, padx=5, pady=5)

    # Row 1
    row1 = ttk.Frame(pref_frame)
    row1.pack(fill=tk.X, padx=5, pady=2)

    ttk.Label(row1, text="Preferred Type:").pack(side=tk.LEFT, padx=5)
    self.type_var = tk.StringVar()
    type_combo = ttk.Combobox(row1, textvariable=self.type_var, width=20)
    type_combo.pack(side=tk.LEFT, padx=5)

    ttk.Label(row1, text="Budget Level:").pack(side=tk.LEFT, padx=5)
    self.budget_var = tk.StringVar()
    budget_combo = ttk.Combobox(row1, textvariable=self.budget_var, width=15,
                                values=["Low", "Medium", "High"])
    budget_combo.pack(side=tk.LEFT, padx=5)

    # Row 2
    row2 = ttk.Frame(pref_frame)
    row2.pack(fill=tk.X, padx=5, pady=2)

    ttk.Label(row2, text="Preferred Region:").pack(side=tk.LEFT, padx=5)
    self.region_var = tk.StringVar()
    region_combo = ttk.Combobox(row2, textvariable=self.region_var, width=20)
    region_combo.pack(side=tk.LEFT, padx=5)
```

```python
    # Row 2
    row2 = ttk.Frame(pref_frame)
    row2.pack(fill=tk.X, padx=5, pady=2)

    ttk.Label(row2, text="Preferred Region:").pack(side=tk.LEFT, padx=5)
    self.region_var = tk.StringVar()
    region_combo = ttk.Combobox(row2, textvariable=self.region_var, width=20)
    region_combo.pack(side=tk.LEFT, padx=5)

    ttk.Button(row2, text="Save Preferences", command=self.save_preferences).pack(side=tk.LEFT, padx=5)
    ttk.Button(row2, text="Load Preferences", command=self.load_preferences).pack(side=tk.LEFT, padx=5)
    ttk.Button(row2, text="Generate Recommendations", command=self.generate_recommendations).pack(side=tk.LEFT, padx=5)

    # Update combo boxes with actual data
    self.update_preference_combos()

    # Recommendations display
    rec_frame = ttk.LabelFrame(frame, text="Personalized Recommendations")
    rec_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

    # Create treeview with scrollbar
    tree_frame = ttk.Frame(rec_frame)
    tree_frame.pack(fill=tk.BOTH, expand=True)

    columns = ("Name", "Type", "Region", "Budget", "Actual Rating", "Predicted Rating")
    self.rec_tree = ttk.Treeview(tree_frame, columns=columns, show="headings", height=12)

    column_widths = {"Name": 150, "Type": 120, "Region": 100, "Budget": 80, "Actual Rating": 100, "Predicted Rating": 120}
    for col in columns:
        self.rec_tree.heading(col, text=col)
        self.rec_tree.column(col, width=column_widths.get(col, 100))
```

The code creates a comprehensive recommendations tab with a user preferences section and a results display area. The preferences panel includes two organized rows of input controls: the first row contains dropdown selectors for travel type and budget level, while the second row features a region selector alongside functional buttons for saving/loading preferences and generating recommendations. Below

this, a dedicated display area shows personalized recommendations using a Treeview widget with six columns presenting key destination information including name, type, region, budget, actual rating, and predicted rating. The interface is designed with proper spacing, consistent styling, and responsive layout management to ensure optimal user experience.

```python
    def create_visualization_tab(self):
        """Create visualization tab"""
        frame = ttk.Frame(self.notebook)
        self.notebook.add(frame, text="Data Analysis")

        # Control buttons
        btn_frame = ttk.Frame(frame)
        btn_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Button(btn_frame, text="Show Predicted Ratings",
                   command=self.show_predicted_ratings).pack(side=tk.LEFT, padx=5)
        ttk.Button(btn_frame, text="Show Ratings Comparison",
                   command=self.show_ratings_comparison).pack(side=tk.LEFT, padx=5)

        # Canvas frame
        self.canvas_frame = ttk.Frame(frame)
        self.canvas_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

```

The code creates a dedicated visualization tab labeled "Data Analysis" within the application's notebook interface. It features a control panel at the top containing buttons for generating different types of visualizations, including "Show Predicted Ratings" and "Show Ratings Comparison". Below the control buttons, a canvas frame is established to serve as a container for displaying the generated charts and graphs, utilizing flexible layout management to ensure the visualizations properly fill the available space with appropriate padding.

```python
    def update_preference_combos(self):
        """Update preference combo boxes with actual data"""
        destinations = self.system.get_all_destinations()

        # Get unique types and regions
        types = sorted(list(set(d['type'] for d in destinations)))
        regions = sorted(list(set(d.get('region', '') for d in destinations if d.get('region'))))

        # Update combobox values
        self.type_var.set('')
        type_combo = self.get_widget_by_var(self.type_var)
        if type_combo:
            type_combo['values'] = types

        self.region_var.set('')
        region_combo = self.get_widget_by_var(self.region_var)
        if region_combo:
            region_combo['values'] = regions

    def get_widget_by_var(self, var):
        """Helper function to find widget by variable"""
        for widget in self.root.winfo_children():
            if hasattr(widget, 'tkvar') and widget.tkvar == var:
                return widget
        return None

```

The `update_preference_combos` function dynamically populates the preference dropdown menus with actual destination data from the system. It extracts unique

travel types and regions from all available destinations, sorts them alphabetically, and updates the corresponding combo boxes. The function clears any existing selections and ensures the dropdown values match the current dataset. A helper function `get_widget_by_var` supports this process by locating the specific widget associated with each variable, enabling precise updates to the user interface components.

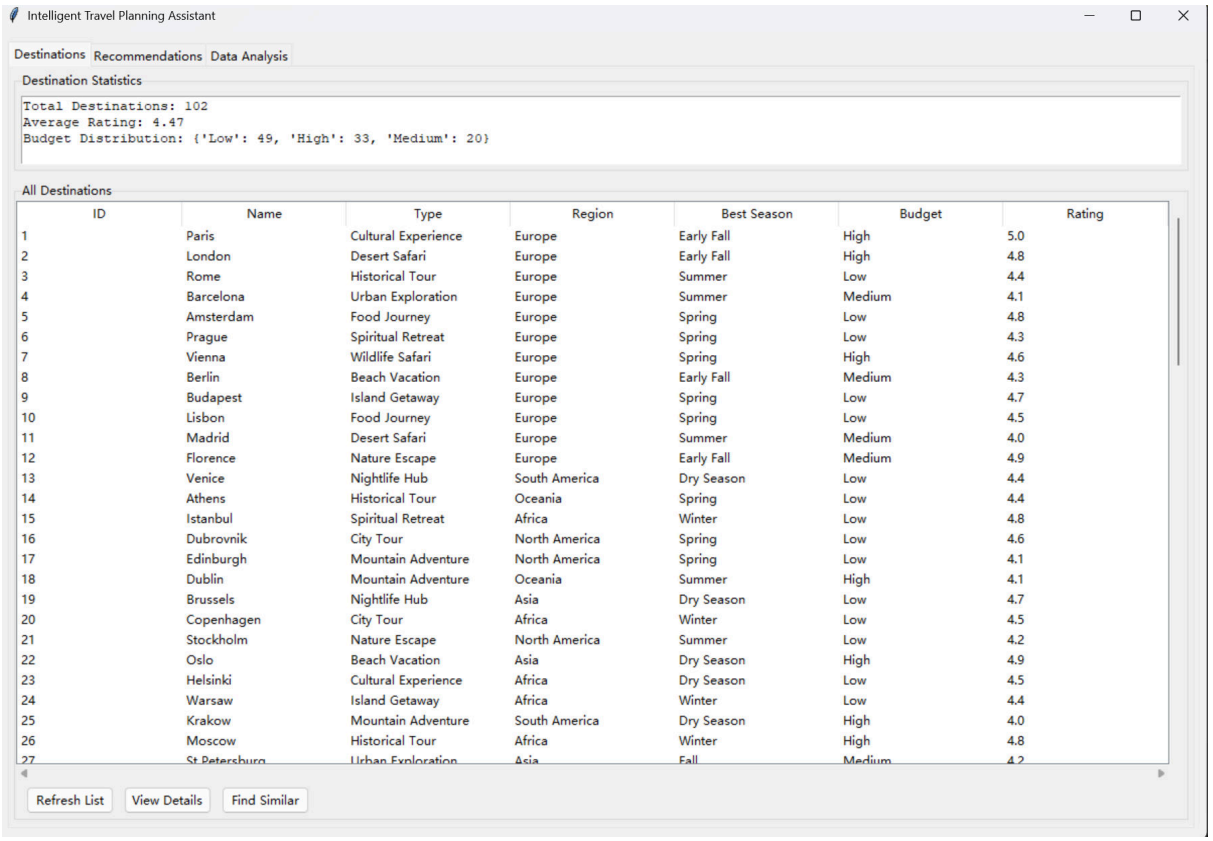
```
    def show_similar_destinations(self):
        """Show similar destinations"""
        selection = self.dest_tree.selection()
        if not selection:
            messagebox.showwarning("Warning", "Please select a destination first!")
            return

        item = self.dest_tree.item(selection[0])
        dest_id = item['values'][0]

        similar_destinations = self.system.get_similar_recommendations(dest_id)

        if similar_destinations:
            details = "Similar Destinations:\n" + "="*30 + "\n"
            for dest in similar_destinations:
                details += f"\nName: {dest['name']}\n"
                details += f"Type: {dest['type']}\n"
                details += f"Similarity Score: {dest.get('similarity_score', 0):.3f}\n"
                details += f"Rating: {dest['rating']}\n"
                details += "-" * 20 + "\n"

            messagebox.showinfo("Similar Destinations", details)
        else:
            messagebox.showinfo("Similar Destinations", "No similar destinations found.")

```

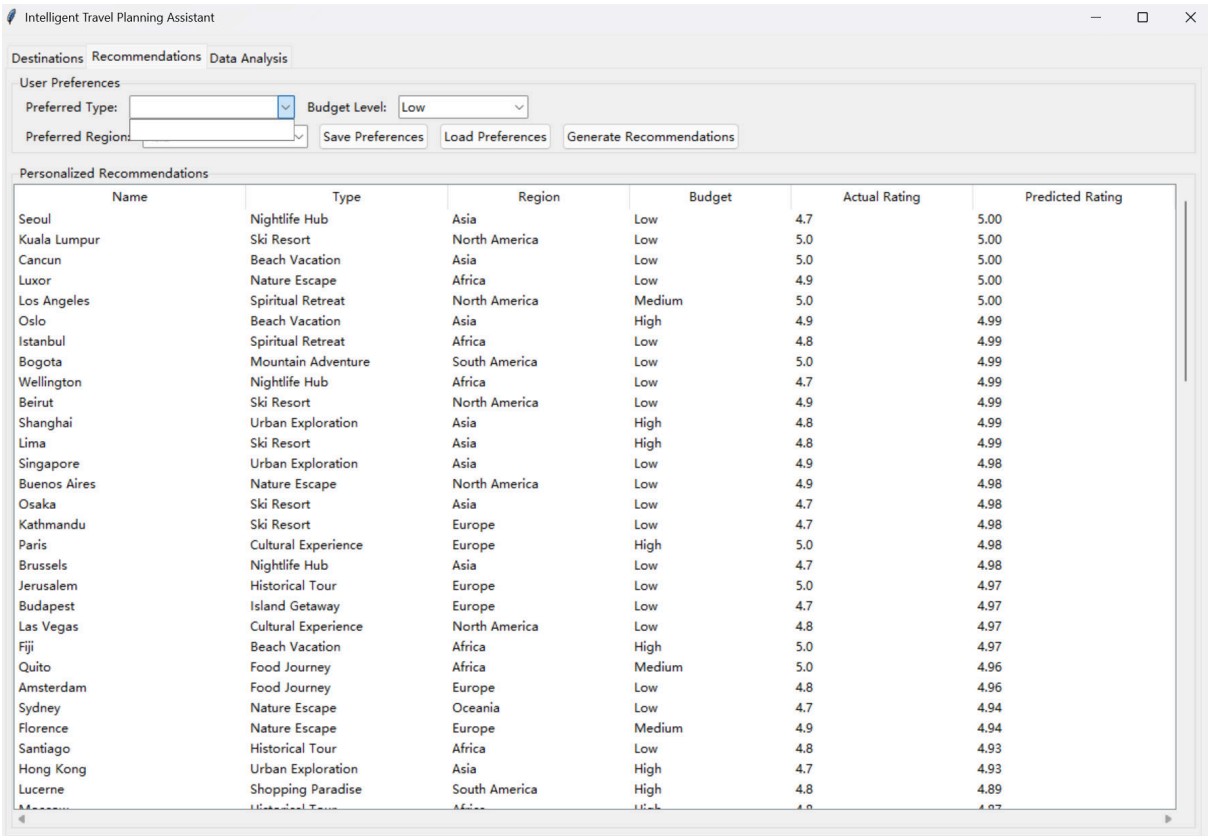
The `show_similar_destinations` function enables users to find destinations similar to one they've selected. When triggered, it first checks if a destination has been selected from the list. If no selection is made, it displays a warning message. For a valid selection, it extracts the destination ID and queries the system for similar recommendations. If similar destinations are found, it formats a detailed display showing each destination's name, type, similarity score, and rating, presenting this information in a structured message box. If no similar destinations are available, it informs the user accordingly.

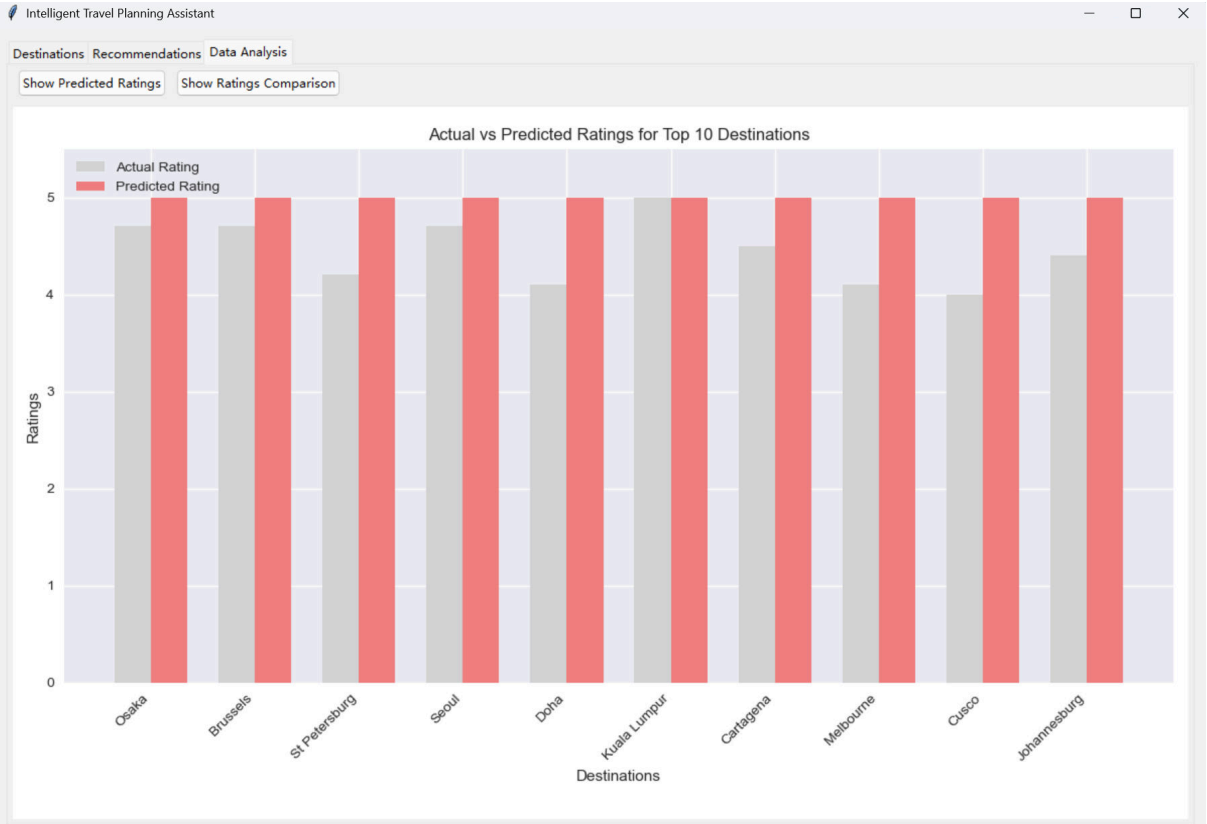
```python
class Visualization:
    def __init__(self):
        plt.style.use('seaborn-v0_8')
        self.fig_size = (12, 8)

    def plot_predicted_ratings(self, df: pd.DataFrame):
        """Plot predicted ratings"""
        if 'predicted_rating' not in df.columns or df.empty:
            return None

        top_15 = df.head(15)

        fig, ax = plt.subplots(figsize=(12, 8))
        y_pos = np.arange(len(top_15))

        bars = ax.barh(y_pos, top_15['predicted_rating'], color='lightblue', alpha=0.7)
        ax.set_yticks(y_pos)
        ax.set_yticklabels(top_15['name'])
        ax.set_xlabel('Predicted Rating')
        ax.set_title('Top 15 Recommended Destinations - Predicted Ratings')
        ax.set_xlim(0, 5.5)

        # Add value labels
        for i, (bar, rating) in enumerate(zip(bars, top_15['predicted_rating'])):
            ax.text(bar.get_width() + 0.1, bar.get_y() + bar.get_height()/2,
                    f'{rating:.2f}', va='center', fontweight='bold')

        plt.tight_layout()
        return fig
```

```python
    def plot_ratings_comparison(self, df: pd.DataFrame):
        """Plot comparison between actual and predicted ratings"""
        if 'predicted_rating' not in df.columns or df.empty:
            return None

        top_10 = df.head(10)

        fig, ax = plt.subplots(figsize=(12, 6))
        x_pos = np.arange(len(top_10))
        width = 0.35

        bars1 = ax.bar(x_pos - width/2, top_10['rating'], width, label='Actual Rating', color='lightgray')
        bars2 = ax.bar(x_pos + width/2, top_10['predicted_rating'], width, label='Predicted Rating', color='lightcoral')

        ax.set_xlabel('Destinations')
        ax.set_ylabel('Ratings')
        ax.set_title('Actual vs Predicted Ratings for Top 10 Destinations')
        ax.set_xticks(x_pos)
        ax.set_xticklabels(top_10['name'], rotation=45, ha='right')
        ax.legend()
        ax.set_ylim(0, 5.5)

        plt.tight_layout()
        return fig
```

The `Visualization` class contains two methods for creating comparative charts. The `plot_predicted_ratings` method generates a horizontal bar chart displaying the top 15 recommended destinations with their predicted ratings, featuring value labels and a clean seaborn style. The `plot_ratings_comparison` method creates a side-by-side bar chart comparing actual versus predicted ratings for the top 10 destinations, using different colors to distinguish between the two rating types and including proper axis labels, legends, and rotated destination names for readability. Both methods return matplotlib figure objects for display in the application's interface.


Intelligent Travel Planning Assistant
Destinations
Recommendations
Data Analysis
Destination Statistics
Total Destinations: 102
Average Rating: 4.47
Budget Distribution: {'Low': 49, 'High': 33, 'Medium': 20}
All Destinations
ID Name Type Region Best Season Budget Rating
1 Paris Cultural Experience Europe Early Fall High 5.0
2 London Desert Safari Europe Early Fall High 4.8
3 Rome Historical Tour Europe Summer Low 4.4
4 Barcelona Urban Exploration Europe Summer Medium 4.1
5 Amsterdam Food Journey Europe Spring Low 4.8
6 Prague Spiritual Retreat Europe Spring Low 4.3
7 Vienna Wildlife Safari Europe Spring High 4.6
8 Berlin Beach Vacation Europe Early Fall Medium 4.3
9 Budapest Island Getaway Europe Spring Low 4.7
10 Lisbon Food Journey Europe Spring Low 4.5
11 Madrid Desert Safari Europe Summer Medium 4.0
12 Florence Nature Escape Europe Early Fall Medium 4.9
13 Venice Nightlife Hub South America Dry Season Low 4.4
14 Athens Historical Tour Oceania Spring Low 4.4
15 Istanbul Spiritual Retreat Africa Winter Low 4.8
16 Dubrovnik City Tour North America Spring Low 4.6
17 Edinburgh Mountain Adventure North America Spring Low 4.1
18 Dublin Mountain Adventure Oceania Summer High 4.1
19 Brussels Nightlife Hub Asia Dry Season Low 4.7
20 Copenhagen City Tour Africa Winter Low 4.5
21 Stockholm Nature Escape North America Summer Low 4.2
22 Oslo Beach Vacation Asia Dry Season High 4.9
23 Helsinki Cultural Experience Africa Dry Season Low 4.5
24 Warsaw Island Getaway Africa Winter Low 4.4
25 Krakow Mountain Adventure South America Dry Season High 4.0
26 Moscow Historical Tour Africa Winter High 4.8
Refresh List
View Details
Find Similar


Intelligent Travel Planning Assistant
Destinations
Recommendations
Data Analysis
User Preferences
Preferred Type:
Budget Level: Low
Preferred Region:
Save Preferences
Load Preferences
Generate Recommendations
Personalized Recommendations
Name Type Region Budget Actual Rating Predicted Rating
Seoul Nightlife Hub Asia Low 4.7 5.00
Kuala Lumpur Ski Resort North America Low 5.0 5.00
Cancun Beach Vacation Asia Low 5.0 5.00
Luxor Nature Escape Africa Low 4.9 5.00
Los Angeles Spiritual Retreat North America Medium 5.0 5.00
Oslo Beach Vacation Asia High 4.9 4.99
Istanbul Spiritual Retreat Africa Low 4.8 4.99
Bogota Mountain Adventure South America Low 5.0 4.99
Wellington Nightlife Hub Africa Low 4.7 4.99
Beirut Ski Resort North America Low 4.9 4.99
Shanghai Urban Exploration Asia High 4.8 4.99
Lima Ski Resort Asia High 4.8 4.99
Singapore Urban Exploration Asia Low 4.9 4.98
Buenos Aires Nature Escape North America Low 4.9 4.98
Osaka Ski Resort Asia Low 4.7 4.98
Kathmandu Ski Resort Europe Low 4.7 4.98
Paris Cultural Experience Europe High 5.0 4.98
Brussels Nightlife Hub Asia Low 4.7 4.98
Jerusalem Historical Tour Europe Low 5.0 4.97
Budapest Island Getaway Europe Low 4.7 4.97
Las Vegas Cultural Experience North America Low 4.8 4.97
Fiji Beach Vacation Africa High 5.0 4.97
Quito Food Journey Africa Medium 5.0 4.96
Amsterdam Food Journey Europe Low 4.8 4.96
Sydney Nature Escape Oceania Low 4.7 4.94
Florence Nature Escape Europe Medium 4.9 4.94
Santiago Historical Tour Africa Low 4.8 4.93
Hong Kong Urban Exploration Asia High 4.7 4.93
Lucerne Shopping Paradise South America High 4.8 4.89


Intelligent Travel Planning Assistant
Destinations
Recommendations
Data Analysis
Show Predicted Ratings
Show Ratings Comparison
Actual vs Predicted Ratings for Top 10 Destinations
Actual Rating
Predicted Rating
Ratings
0
1
2
3
4
5
Osaka
Brussels
St Petersburg
Seoul
Doha
Kuala Lumpur
Cartagena
Melbourne
Cusco
Johannesburg
Destinations

Project Wrap Up (D4)

Progress in Wrapping Up the Project

To build this project, and extend my programming capabilities, I did the following:

<date> - <describe action>
<provide a screenshot image or notes of what **you did/created** related to this>
<describe what you learnt from this>

Distinction Achievement

Impressive - high quality code

Functional and Level of Completion

Data Organisation

Functional Decomposition - use of functions, procedures, methods

Build and Run Instructions

Demo Video Link

High Distinction Achievement (H2)

Other Libraries / Language Features / Tools Used

Degree of Ownership and Future Direction

Efficient Core Architecture

Creativity and sophistication

References

Generative AI acknowledgement

This file has additional line breaks applied by OnTrack because they contain lines longer than the configured limit. Lines over 1000 characters long have been truncated to limit PDF page count. The orginal submission can be retrieved via the "Download Uploaded Files" function.

```
import tkinter as tk
from tkinter import ttk, messagebox, scrolledtext
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from travel_system import TravelPlanningSystem

class TravelPlannerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Intelligent Travel Planning Assistant")
        self.root.geometry("1200x800")

        self.system = TravelPlanningSystem()

        # Create main frame
        self.main_frame = ttk.Frame(root)
        self.main_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Create notebook
        self.notebook = ttk.Notebook(self.main_frame)
        self.notebook.pack(fill=tk.BOTH, expand=True)

        self.create_destinations_tab()
        self.create_recommendations_tab()
        self.create_visualization_tab()

        # Load initial data
        self.refresh_destinations()

    def create_destinations_tab(self):
        """Create destinations management tab"""
        frame = ttk.Frame(self.notebook)
        self.notebook.add(frame, text="Destinations")

        # Statistics frame
        stats_frame = ttk.LabelFrame(frame, text="Destination Statistics")
        stats_frame.pack(fill=tk.X, padx=5, pady=5)

        self.stats_text = tk.Text(stats_frame, height=4, wrap=tk.WORD)
        self.stats_text.pack(fill=tk.X, padx=5, pady=5)

        # Destinations list
        list_frame = ttk.LabelFrame(frame, text="All Destinations")
        list_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        # Create treeview with scrollbar
        tree_frame = ttk.Frame(list_frame)
        tree_frame.pack(fill=tk.BOTH, expand=True)

```

```
        columns = ("ID", "Name", "Type", "Region", "Best Season", "Budget",
        ↪  "Rating")
        self.dest_tree = ttk.Treeview(tree_frame, columns=columns, show="headings",
        ↪  height=15)

        for col in columns:
            self.dest_tree.heading(col, text=col)
            self.dest_tree.column(col, width=100)

        # Scrollbars
        v_scrollbar = ttk.Scrollbar(tree_frame, orient=tk.VERTICAL,
        ↪  command=self.dest_tree.yview)
        h_scrollbar = ttk.Scrollbar(tree_frame, orient=tk.HORIZONTAL,
        ↪  command=self.dest_tree.xview)
        self.dest_tree.configure(yscrollcommand=v_scrollbar.set,
        ↪  xscrollcommand=h_scrollbar.set)

        self.dest_tree.grid(row=0, column=0, sticky='nsew')
        v_scrollbar.grid(row=0, column=1, sticky='ns')
        h_scrollbar.grid(row=1, column=0, sticky='ew')

        tree_frame.grid_rowconfigure(0, weight=1)
        tree_frame.grid_columnconfigure(0, weight=1)

        # Button frame
        btn_frame = ttk.Frame(list_frame)
        btn_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Button(btn_frame, text="Refresh List",
        ↪  command=self.refresh_destinations).pack(side=tk.LEFT, padx=5)
        ttk.Button(btn_frame, text="View Details",
        ↪  command=self.show_destination_details).pack(side=tk.LEFT, padx=5)
        ttk.Button(btn_frame, text="Find Similar",
        ↪  command=self.show_similar_destinations).pack(side=tk.LEFT, padx=5)

    def create_recommendations_tab(self):
        """Create recommendations tab"""
        frame = ttk.Frame(self.notebook)
        self.notebook.add(frame, text="Recommendations")

        # User preferences
        pref_frame = ttk.LabelFrame(frame, text="User Preferences")
        pref_frame.pack(fill=tk.X, padx=5, pady=5)

        # Row 1
        row1 = ttk.Frame(pref_frame)
        row1.pack(fill=tk.X, padx=5, pady=2)

        ttk.Label(row1, text="Preferred Type:").pack(side=tk.LEFT, padx=5)
        self.type_var = tk.StringVar()
        type_combo = ttk.Combobox(row1, textvariable=self.type_var, width=28)
        type_combo.pack(side=tk.LEFT, padx=5)

        ttk.Label(row1, text="Budget Level:").pack(side=tk.LEFT, padx=5)
```

```
        self.budget_var = tk.StringVar()
        budget_combo = ttk.Combobox(row1, textvariable=self.budget_var, width=15,
            values=["Low", "Medium", "High"])
        budget_combo.pack(side=tk.LEFT, padx=5)

        # Row 2
        row2 = ttk.Frame(pref_frame)
        row2.pack(fill=tk.X, padx=5, pady=2)

        ttk.Label(row2, text="Preferred Region:").pack(side=tk.LEFT, padx=5)
        self.region_var = tk.StringVar()
        region_combo = ttk.Combobox(row2, textvariable=self.region_var, width=28)
        region_combo.pack(side=tk.LEFT, padx=5)

        ttk.Button(row2, text="Save Preferences",
        ↪  command=self.save_preferences).pack(side=tk.LEFT, padx=5)
        ttk.Button(row2, text="Load Preferences",
        ↪  command=self.load_preferences).pack(side=tk.LEFT, padx=5)
        ttk.Button(row2, text="Generate Recommendations",
        ↪  command=self.generate_recommendations).pack(side=tk.LEFT, padx=5)

        # Update combo boxes with actual data
        self.update_preference_combos()

        # Recommendations display
        rec_frame = ttk.LabelFrame(frame, text="Personalized Recommendations")
        rec_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        # Create treeview with scrollbar
        tree_frame = ttk.Frame(rec_frame)
        tree_frame.pack(fill=tk.BOTH, expand=True)

        columns = ("Name", "Type", "Region", "Budget", "Actual Rating", "Predicted
        ↪  Rating")
        self.rec_tree = ttk.Treeview(tree_frame, columns=columns, show="headings",
        ↪  height=12)

        column_widths = {"Name": 150, "Type": 120, "Region": 100, "Budget": 80,
        ↪  "Actual Rating": 100, "Predicted Rating": 120}
        for col in columns:
            self.rec_tree.heading(col, text=col)
            self.rec_tree.column(col, width=column_widths.get(col, 100))

        # Scrollbars
        v_scrollbar = ttk.Scrollbar(tree_frame, orient=tk.VERTICAL,
        ↪  command=self.rec_tree.yview)
        h_scrollbar = ttk.Scrollbar(tree_frame, orient=tk.HORIZONTAL,
        ↪  command=self.rec_tree.xview)
        self.rec_tree.configure(yscrollcommand=v_scrollbar.set,
        ↪  xscrollcommand=h_scrollbar.set)

        self.rec_tree.grid(row=0, column=0, sticky='nsew')
        v_scrollbar.grid(row=0, column=1, sticky='ns')
        h_scrollbar.grid(row=1, column=0, sticky='ew')
```

```
        tree_frame.grid_rowconfigure(0, weight=1)
        tree_frame.grid_columnconfigure(0, weight=1)

    def create_visualization_tab(self):
        """Create visualization tab"""
        frame = ttk.Frame(self.notebook)
        self.notebook.add(frame, text="Data Analysis")

        # Control buttons
        btn_frame = ttk.Frame(frame)
        btn_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Button(btn_frame, text="Show Predicted Ratings",
                   command=self.show_predicted_ratings).pack(side=tk.LEFT, padx=5)
        ttk.Button(btn_frame, text="Show Ratings Comparison",
                   command=self.show_ratings_comparison).pack(side=tk.LEFT, padx=5)

        # Canvas frame
        self.canvas_frame = ttk.Frame(frame)
        self.canvas_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

    def update_preference_combos(self):
        """Update preference combo boxes with actual data"""
        destinations = self.system.get_all_destinations()

        # Get unique types and regions
        types = sorted(list(set(d['type'] for d in destinations)))
        regions = sorted(list(set(d.get('region', '') for d in destinations if
        ↪  d.get('region'))))

        # Update combobox values
        self.type_var.set('')
        type_combo = self.get_widget_by_var(self.type_var)
        if type_combo:
            type_combo['values'] = types

        self.region_var.set('')
        region_combo = self.get_widget_by_var(self.region_var)
        if region_combo:
            region_combo['values'] = regions

    def get_widget_by_var(self, var):
        """Helper function to find widget by variable"""
        for widget in self.root.winfo_children():
            if hasattr(widget, 'tkvar') and widget.tkvar == var:
                return widget
        return None

    def refresh_destinations(self):
        """Refresh destinations list and statistics"""
        # Clear existing items
        for item in self.dest_tree.get_children():
            self.dest_tree.delete(item)
```

```
194
195         destinations = self.system.get_all_destinations()
196
197         # Update statistics
198         stats = self.system.get_destination_statistics()
199         self.stats_text.delete(1.0, tk.END)
200         if stats:
201             self.stats_text.insert(tk.END, f"Total Destinations:
                ↪  {stats['total_destinations']}\n")
202             self.stats_text.insert(tk.END, f"Average Rating:
                ↪  {stats['average_rating']:.2f}\n")
203             self.stats_text.insert(tk.END, f"Budget Distribution:
                ↪  {stats['budget_distribution']}\n")
204
205         # Add destinations to treeview
206         for dest in destinations:
207             self.dest_tree.insert("", tk.END, values=(
208                 dest['id'],
209                 dest['name'],
210                 dest['type'],
211                 dest.get('region', 'N/A'),
212                 dest['best_season'],
213                 dest['budget_level'],
214                 dest['rating']
215             ))
216
217         # Update preference combos
218         self.update_preference_combos()
219
220     def show_destination_details(self):
221         """Show destination details"""
222         selection = self.dest_tree.selection()
223         if not selection:
224             messagebox.showwarning("Warning", "Please select a destination!")
225             return
226
227         item = self.dest_tree.item(selection[0])
228         dest_id = item['values'][0]
229
230         destination = self.system.get_destination_by_id(dest_id)
231         if destination:
232             details = f"Destination Details:\n{'='*40}\n"
233             details += f"Name: {destination['name']}\n"
234             details += f"Type: {destination['type']}\n"
235             details += f"Region: {destination.get('region', 'N/A')}\n"
236             details += f"Best Season: {destination['best_season']}\n"
237             details += f"Budget Level: {destination['budget_level']}\n"
238             details += f"Climate: {destination['climate']}\n"
239             details += f"Rating: {destination['rating']}\n"
240             details += f"Safety Level: {destination.get('safety_level', 'N/A')}\n"
241             details += f"Adventure Level: {destination.get('adventure_level',
                ↪  'N/A')}\n"
242             details += f"Attractions: {', '.join(destination['attractions'])}\n"
243             details += f"Specialties: {', '.join(destination['specialty'])}\n"
```

```
            details += f"Tags: {', '.join(destination['tags'])}\n"
            details += f"Ideal Duration: {destination.get('ideal_duration',
            ↪  'N/A')}\n"
            details += f"Language: {destination.get('language', 'N/A')}\n"
            details += f"Currency: {destination.get('currency', 'N/A')}\n"

            messagebox.showinfo("Destination Details", details)

    def show_similar_destinations(self):
        """Show similar destinations"""
        selection = self.dest_tree.selection()
        if not selection:
            messagebox.showwarning("Warning", "Please select a destination first!")
            return

        item = self.dest_tree.item(selection[0])
        dest_id = item['values'][0]

        similar_destinations = self.system.get_similar_recommendations(dest_id)

        if similar_destinations:
            details = "Similar Destinations:\n" + "="*30 + "\n"
            for dest in similar_destinations:
                details += f"\nName: {dest['name']}\n"
                details += f"Type: {dest['type']}\n"
                details += f"Similarity Score: {dest.get('similarity_score',
                ↪  0):.3f}\n"
                details += f"Rating: {dest['rating']}\n"
                details += "-" * 20 + "\n"

            messagebox.showinfo("Similar Destinations", details)
        else:
            messagebox.showinfo("Similar Destinations", "No similar destinations
            ↪  found.")

    def save_preferences(self):
        """Save user preferences and retrain model"""
        preferences = {
            'preferred_type': self.type_var.get(),
            'budget_level': self.budget_var.get(),
            'preferred_region': self.region_var.get()
        }
        self.system.update_user_preferences(preferences)

        messagebox.showinfo("Success", "User preferences saved!")

    def load_preferences(self):
        """Load user preferences"""
        preferences = self.system.get_user_preferences()
        self.type_var.set(preferences.get('preferred_type', ''))
        self.budget_var.set(preferences.get('budget_level', ''))
        self.region_var.set(preferences.get('preferred_region', ''))
        messagebox.showinfo("Success", "User preferences loaded!")

```

```
295     def generate_recommendations(self):
296         """Generate recommendations"""
297         for item in self.rec_tree.get_children():
298             self.rec_tree.delete(item)
299 
300         recommendations = self.system.get_personalized_recommendations()
301         if not recommendations.empty:
302             for _, dest in recommendations.iterrows():
303                 self.rec_tree.insert("", tk.END, values=(
304                     dest['name'],
305                     dest['type'],
306                     dest.get('region', 'N/A'),
307                     dest['budget_level'],
308                     dest['rating'],
309                     f"{dest.get('predicted_rating', 0):.2f}"
310                 ))
311         else:
312             messagebox.showinfo("Info", "No recommendations available. Please set
                ↪ your preferences first.")
313 
314     def show_predicted_ratings(self):
315         """Show predicted ratings visualization"""
316         self.clear_canvas()
317         recommendations = self.system.get_personalized_recommendations()
318         fig = self.system.visualization.plot_predicted_ratings(recommendations)
319 
320         if fig:
321             canvas = FigureCanvasTkAgg(fig, self.canvas_frame)
322             canvas.draw()
323             canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)
324         else:
325             messagebox.showwarning("Warning", "No recommendation data available.
                ↪ Please generate recommendations first.")
326 
327     def show_ratings_comparison(self):
328         """Show ratings comparison visualization"""
329         self.clear_canvas()
330         recommendations = self.system.get_personalized_recommendations()
331         fig = self.system.visualization.plot_ratings_comparison(recommendations)
332 
333         if fig:
334             canvas = FigureCanvasTkAgg(fig, self.canvas_frame)
335             canvas.draw()
336             canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)
337         else:
338             messagebox.showwarning("Warning", "No recommendation data available.
                ↪ Please generate recommendations first.")
339 
340     def clear_canvas(self):
341         """Clear canvas"""
342         for widget in self.canvas_frame.winfo_children():
343             widget.destroy()
344 
345 if __name__ == "__main__":
```

```
    root = tk.Tk()
    app = TravelPlannerApp(root)
    root.mainloop()
```

20 Custom Program - Final

This is the final custom program task, your opportunity to show what you have been capable of producing.

Outcome	Description
TLO1	Meet the requirements of a D/HD custom project.

Date	Author	Comment
2025/08/06 20:27	Ziyue Meng	Planned date adjusted to 17 Oct.
2025/10/16 14:54	Ziyue Meng	Working On It
2025/10/24 20:04	Ziyue Meng	Assess in Portfolio

Monash

Introduction to Programming

OnTrack Submission

Custom Program - Final

Submitted By:
Ziyue Meng
zmen0034
2025/10/24 20:04

Tutor:
Ganesh Krishnasamy

Task Outcomes		Supports
TLO1	Meet the requirements of a D/HD custom project.	

October 24, 2025

Custom Project

Name: ZIyue Meng
Student ID: 36035432
Date: 5/10/2025

Project Pitch (D2)

Programming Language(s): Python
Using GenAI: Copilot
Target Grade:HD

Overview:

This project develops an intelligent travel planning assistant that provides personalized destination recommendations using machine learning algorithms. The system processes a comprehensive dataset of 100 international travel destinations collected from online sources, offering users a wide selection of global travel options.

The application features a user-friendly graphical interface with three main components: a destination browser for exploring travel locations, a recommendation system that suggests destinations based on user preferences, and basic data visualization for understanding recommendation results. Users can set their travel preferences including destination type, budget level, and preferred regions to receive customized suggestions.

The core functionality is built using Python's scikit-learn library, implementing K-Nearest Neighbors algorithm for finding similar destinations and Random Forest regression for predicting user ratings. The system handles data management through pandas and presents results through an intuitive Tkinter-based interface.

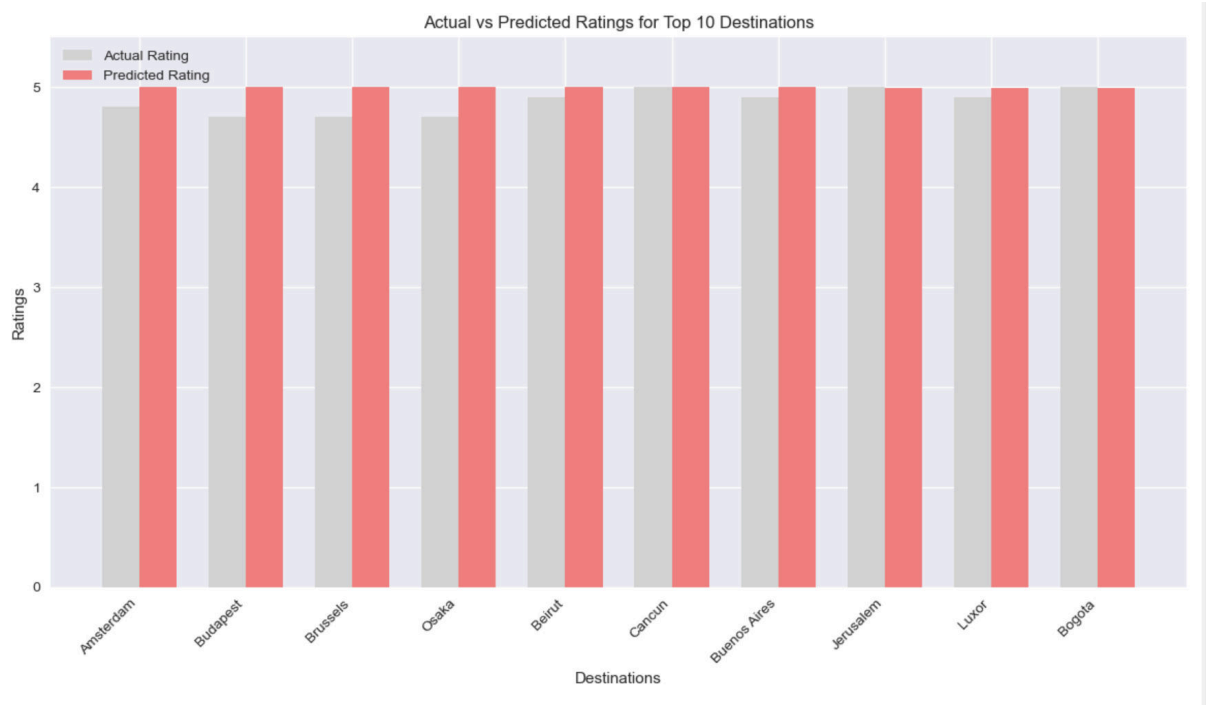
visualization.py > ...

```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from typing import List, Dict
import numpy as np

class Visualization:
    def __init__(self):
        plt.style.use('seaborn-v0_8')
        self.fig_size = (12, 8)

    def plot_predicted_ratings(self, df: pd.DataFrame):
        """Plot predicted ratings"""
        if 'predicted_rating' not in df.columns or df.empty:
            return None

        top_15 = df.head(15)

        fig, ax = plt.subplots(figsize=(12, 8))
        y_pos = np.arange(len(top_15))

        bars = ax.barh(y_pos, top_15['predicted_rating'], color='lightblue', alpha=0.7)
        ax.set_yticks(y_pos)
        ax.set_yticklabels(top_15['name'])
        ax.set_xlabel('Predicted Rating')
        ax.set_title('Top 15 Recommended Destinations - Predicted Ratings')
        ax.set_xlim(0, 5.5)

        # Add value labels
        for i, (bar, rating) in enumerate(zip(bars, top_15['predicted_rating'])):
            ax.text(bar.get_width() + 0.1, bar.get_y() + bar.get_height()/2,
                    f'{rating:.2f}', va='center', fontweight='bold')

        plt.tight_layout()
        return fig

    def plot_ratings_comparison(self, df: pd.DataFrame):
        """Plot comparison between actual and predicted ratings"""
        if 'predicted_rating' not in df.columns or df.empty:
            return None
```

OUTPUT PORTS bash

ents/project_english

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.13.2



main.py > ...

```python
import tkinter as tk
from tkinter import ttk, messagebox, scrolledtext
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from travel_system import TravelPlanningSystem

class TravelPlannerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Intelligent Travel Planning Assistant")
        self.root.geometry("1200x800")

        self.system = TravelPlanningSystem()

        # Create main frame
        self.main_frame = ttk.Frame(root)
        self.main_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Create notebook
        self.notebook = ttk.Notebook(self.main_frame)
        self.notebook.pack(fill=tk.BOTH, expand=True)

        self.create_destinations_tab()
        self.create_recommendations_tab()
        self.create_visualization_tab()

        # Load initial data
        self.refresh_destinations()
```

HD Extension Ideas:

- Automated Data Integration
  - Implement web scraping to dynamically collect and update destination data from travel websites
  - Develop real-time data updating system with automated data validation
  - Integrate multiple online data sources for comprehensive destination information
- Advanced Machine Learning Features
  - Implement collaborative filtering using matrix factorization for improved recommendations
  - Develop content-based filtering that analyzes destination features and user preferences
  - Create hybrid recommendation system combining multiple ML approaches
  - Implement model performance evaluation and comparison framework
- Enhanced Visualization System
  - Develop interactive data dashboards with multiple chart types
  - Implement geospatial visualization showing destinations on maps
  - Create comparative analysis charts for destination features and ratings
  - Build real-time visualization updates based on user interactions

- Intelligent Recommendation Features
    - Implement recommendation explanation system showing why destinations are suggested
    - Develop multi-criteria recommendation considering season, duration, and activities
    - Create similarity analysis showing how destinations relate to each other
    - Build preference learning system that adapts based on user feedback
- Advanced User Features
    - Implement user profile management with preference history
    - Develop saved recommendations and travel planning functionality
    - Create destination comparison tools with side-by-side feature analysis
    - Build export functionality for recommendations and travel plans
- System Optimization
    - Implement caching system for faster recommendation generation
    - Develop automated data preprocessing and cleaning pipelines
    - Create performance monitoring and optimization features
    - Build modular architecture for easy feature expansion
-

How did you come up with your project pitch?

This project was inspired by the challenge of planning travel in an increasingly complex world with countless destination options. Traditional travel planning often involves sifting through overwhelming amounts of information without personalized guidance. I identified an opportunity to apply machine learning techniques to create a smart recommendation system that understands individual preferences.

The concept combines practical travel needs with data science applications, addressing the real problem of destination choice overload. By focusing on personalized recommendations, the system helps users discover destinations they might otherwise overlook. The use of a comprehensive dataset from online sources ensures the recommendations are based on extensive, real-world travel information.

The HD extensions were designed to push the project beyond basic functionality into a sophisticated travel intelligence platform. These enhancements focus on improving recommendation accuracy, providing deeper insights through advanced visualization, and creating a more engaging user experience through interactive features. The project demonstrates how machine learning can transform everyday tasks like travel planning into intelligent, data-driven experiences.

Project Progress (D3)

Prior Knowledge and Experience

I have a solid foundation in several key aspects of programming. I've mastered the use of pointers and references. Pointers are variables that store memory addresses, which allow for direct memory manipulation and efficient data passing between functions. References, on the other hand, act as an alias for an existing variable, providing a more intuitive way to access and modify data in some cases.

I'm also well - versed in loops. For example, the for loop is useful when you know exactly how many times you want to iterate. It has a concise syntax where you can initialize a counter, set a condition for continuation, and update the counter in each iteration. The while loop is more flexible, as it continues as long as a given condition is true, making it suitable for situations where the number of iterations is not fixed in advance. The do - while loop is similar to the while loop, but it executes the loop body at least once before checking the condition.

Enumerations (enum) are a user - defined data type in programming. They allow you to create a set of named constants. For instance, if you are creating a program about the days of the week, you can define an enum like enum Days {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY}; which makes the code more readable and maintainable.

Regarding structures, I can define a structure to encapsulate related data. For example, a "Student" struct containing name, ID, and score. I'm able to properly access its members, which is crucial for handling complex data entities in programming.

When it comes to arrays, I have a good grasp of their usage. I know how to declare and initialize both one - dimensional and two - dimensional arrays. For a one - dimensional array, it can be done like int numbers[5] = {1, 2, 3, 4, 5}; and for a two - dimensional array, something like int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}} works. I'm familiar with iterating through array elements using loops, which is essential for processing the data they hold. Arrays are great for storing collections of similar data, such as a list of student scores or a series of temperatures. I also understand the importance of being cautious with array index bounds to avoid accessing elements outside the valid range, which can lead to unexpected program behavior.

Iterations and Study to Create Project and Extend Capabilities

To build this project, and extend my programming capabilities, I did the following:

The thought process begins with an analysis of the problem: traditional travel planning tools often lack personalized recommendations, and users can feel overwhelmed when faced with too many choices. To address this, the proposed solution involves building an intelligent recommendation system that utilizes machine

learning algorithms and provides visual analytics to help users make more informed and personalized travel decisions.

Core Module Breakdown

1. data_manager.py

   Data Layer: Handles all data operations

2. travel_system.py

   Business Layer: Processes core logic

3. ml_recommender.py

   Algorithm Layer: Machine learning recommendations

4. visualization.py

   Presentation Layer: Data visualization

5. main.py

   Interface Layer: User interaction

```python
class DataManager:
    def __init__(self):
        self.destinations_file = "destinations.json"
        self.users_file = "users.json"
        self.csv_file = "international_travel_destinations.csv"
        self.init_data()

    def init_data(self):
        """Initialize data from CSV file"""
        if os.path.exists(self.csv_file):
            self.load_from_csv()
        else:
            self.init_default_data()
```

The code initializes a `DataManager` class designed to handle multiple data sources, including JSON files for destinations and users, as well as a CSV file for international travel destinations. It implements a flexible data loading strategy by first checking for the existence of the CSV file. If available, it prioritizes loading data from this CSV; if not, it falls back to initializing default data, ensuring system adaptability and robustness.

```python
    def load_from_csv(self):
        """Load destinations from CSV file"""
        try:
            df = pd.read_csv(self.csv_file)
            destinations = []

            for _, row in df.iterrows():
                destination = {
                    "id": int(row['id']),
                    "name": row['name'],
                    "type": row['type'],
                    "attractions": row['attractions'].split(', ') if pd.notna(row['attractions']) else [],
                    "best_season": row['best_season'],
                    "specialty": row['specialties'].split(', ') if pd.notna(row['specialties']) else [],
                    "budget_level": row['budget_level'],
                    "climate": row['climate'],
                    "rating": float(row['rating']),
                    "tags": row['tags'].split(', ') if pd.notna(row['tags']) else [],
                    "region": row.get('region', ''),
                    "ideal_duration": row.get('ideal_duration', ''),
                    "language": row.get('language', ''),
                    "currency": row.get('currency', ''),
                    "safety_level": row.get('safety_level', ''),
                    "family_friendly": bool(row.get('family_friendly', True)),
                    "adventure_level": row.get('adventure_level', ''),
                    "nightlife_rating": row.get('nightlife_rating', ''),
                    "shopping_rating": row.get('shopping_rating', '')
                }
                destinations.append(destination)
```

```python
                self.save_destinations(destinations)
                print(f"Loaded {len(destinations)} destinations from CSV")

            except Exception as e:
                print(f"Error loading CSV: {e}")
                self.init_default_data()
```

The `load_from_csv` function reads and processes data from a CSV file, converting it into the system's required format. It utilizes pandas to load the CSV data into a DataFrame, then iterates through each row to perform data cleaning and format transformation. During this process, it extracts and converts various fields—such as splitting the 'attractions' string into a list—and compiles the processed data into a list of destination dictionaries. If successful, it saves the converted data; if any errors occur during loading or processing, it catches the exception, outputs an error message, and falls back to initializing the system with default data.

```python
class TravelPlanningSystem:
    def __init__(self):
        self.data_manager = DataManager()
        self.ml_recommender = MLRecommender(self.data_manager)
        self.visualization = Visualization()
        self.current_user = "default_user"

    def add_destination(self, destination_data: Dict) -> bool:
        """Add a destination"""
        destinations = self.data_manager.load_destinations()

        # Generate new ID
        new_id = max([d['id'] for d in destinations], default=0) + 1
        destination_data['id'] = new_id

        destinations.append(destination_data)
        self.data_manager.save_destinations(destinations)

        # Retrain models
        self.ml_recommender.train_knn_model()
        return True

    def get_all_destinations(self) -> List[Dict]:
        """Get all destinations"""
        return self.data_manager.load_destinations()
```

The code establishes a `TravelPlanningSystem` class that functions as the central coordinator, integrating all key modules into a complete workflow. During initialization, it creates instances of the `DataManager`, `MLRecommender`, and `Visualization` classes, and sets a default user. Crucially, it includes a bridge method, `get_personalized_recommendations`, which connects the data management and machine learning components. This method retrieves the user's preferences and then leverages the ML recommender to generate and return predicted ratings, demonstrating the system's interconnected architecture.

```python
    def update_user_preferences(self, preferences: Dict):
        """Update user preferences"""
        users_data = self.data_manager.load_users()
        users_data['user_preferences'][self.current_user] = preferences
        self.data_manager.save_users(users_data)

        # Retrain rating prediction model
        self.ml_recommender.train_rating_model(preferences)

    def get_user_preferences(self) -> Dict:
        """Get user preferences"""
        users_data = self.data_manager.load_users()
        return users_data['user_preferences'].get(self.current_user, {})
```

The code implements a user-centric function, `update_user_preferences`, which manages a core input for the recommendation system. When a user's preferences are updated, the system first loads all user data, then specifically updates the current user's preference record before saving the modified data. Crucially, the system is designed to immediately retrain the machine learning model using these new preferences upon any update, ensuring that the recommendations remain dynamically aligned with the user's latest interests.

```python
def prepare_features(self, df: pd.DataFrame) -> pd.DataFrame:
    """Prepare features for machine learning"""
    df_encoded = df.copy()

    # Encode categorical variables
    categorical_columns = ['type', 'best_season', 'budget_level', 'climate']
    for col in categorical_columns:
        if col in df_encoded.columns:
            self.label_encoders[col] = LabelEncoder()
            df_encoded[col + '_encoded'] = self.label_encoders[col].fit_transform(df_encoded[col].fillna('Unknown'))

    # Create tag features (convert tags list to features)
    all_tags = set()
    for tags in df_encoded['tags']:
        if isinstance(tags, list):
            all_tags.update(tags)

    for tag in all_tags:
        df_encoded[f'tag_{tag}'] = df_encoded['tags'].apply(
            lambda x: 1 if isinstance(x, list) and tag in x else 0
        )

    # Select numerical features
    feature_columns = [col for col in df_encoded.columns if col.endswith('_encoded') or col.startswith('tag_')]
    feature_columns.extend(['rating'])

    return df_encoded[feature_columns].fillna(0)
```

The `prepare_features` function performs essential feature engineering to transform raw data into a format suitable for machine learning. It begins by creating a copy of the input DataFrame, then addresses the challenge of converting categorical text data into numerical values. Using Label Encoders, it processes categorical columns like 'type' and 'climate', creating new encoded columns for each.

A key innovation in this process is the transformation of the 'tags' list into a feature matrix. The function first identifies all unique tags across the dataset, then creates binary indicator columns for each tag, marking their presence (1) or absence (0) for

each destination. The final output is a cleaned DataFrame containing only the engineered numerical features, with any missing values filled with zeros.

```
    def train_knn_model(self, n_neighbors=5):
        """Train KNN recommendation model"""
        df = self.data_manager.get_destinations_dataframe()
        if len(df) == 0:
            return

        features = self.prepare_features(df)
        self.knn_model = NearestNeighbors(n_neighbors=min(n_neighbors, len(df)), metric='cosine')
        self.knn_model.fit(features)

    def train_kmeans_model(self, n_clusters=5):
        """Train KMeans clustering model"""
        df = self.data_manager.get_destinations_dataframe()
        if len(df) == 0:
            return

        features = self.prepare_features(df)
        n_clusters = min(n_clusters, len(df))
        self.kmeans_model = KMeans(n_clusters=n_clusters, random_state=42)
        self.kmeans_model.fit(features)
        return self.kmeans_model.labels_

    def train_rating_model(self, user_preferences: Dict):
        """Train rating prediction model"""
        df = self.data_manager.get_destinations_dataframe()
        if len(df) == 0 or not user_preferences:
            return
```

This code defines three distinct machine learning training methods within the recommendation system:

The `train_knn_model` method builds a K-Nearest Neighbors model using cosine similarity to identify similar destinations, automatically adjusting the number of neighbors based on available data size.

The `train_kmeans_model` method performs destination clustering using K-Means algorithm, grouping similar travel spots into clusters for pattern discovery and recommendation diversification.

The `train_rating_model` method prepares a rating prediction model that leverages user preferences to forecast potential ratings for various destinations.

All three methods incorporate safety checks to ensure sufficient data exists before model training, and utilize the feature engineering pipeline to convert raw destination data into machine-readable numerical features.

```python
def train_rating_model(self, user_preferences: Dict):
    """Train rating prediction model"""
    df = self.data_manager.get_destinations_dataframe()
    if len(df) == 0 or not user_preferences:
        return

    # Create simulated ratings based on user preferences
    features = self.prepare_features(df)

    # Simple rule-based rating prediction
    y = df['rating'].copy()

    # Adjust ratings based on user preferences
    for idx, destination in df.iterrows():
        preference_boost = 0
        if user_preferences.get('preferred_type') and user_preferences['preferred_type'] in destination['type']:
            preference_boost += 0.5
        if user_preferences.get('budget_level') and user_preferences['budget_level'] == destination['budget_level']:
            preference_boost += 0.3
        if user_preferences.get('preferred_region') and user_preferences['preferred_region'] in str(destination.get('region', '')):
            preference_boost += 0.2

        y.iloc[idx] = min(5.0, y.iloc[idx] + preference_boost)

    self.rating_model = RandomForestRegressor(n_estimators=20, random_state=42)
    self.rating_model.fit(features, y)
```

The `train_rating_model` function implements a hybrid approach to train a rating prediction model that combines content-based filtering with machine learning. It begins by retrieving destination data and applying feature engineering. The core innovation lies in creating simulated ratings based on user preferences - it starts with existing destination ratings, then applies rule-based adjustments where ratings are boosted when destinations match the user's preferred type, budget level, or region. These preference-based boosts are carefully weighted and capped at the maximum rating of 5.0. Finally, the enhanced ratings are used to train a Random Forest Regressor, creating a personalized recommendation model that learns from both inherent destination features and explicit user preferences.

```python
class TravelPlannerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Intelligent Travel Planning Assistant")
        self.root.geometry("1200x800")

        self.system = TravelPlanningSystem()

        # Create main frame
        self.main_frame = ttk.Frame(root)
        self.main_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Create notebook
        self.notebook = ttk.Notebook(self.main_frame)
        self.notebook.pack(fill=tk.BOTH, expand=True)

        self.create_destinations_tab()
        self.create_recommendations_tab()
        self.create_visualization_tab()

        # Load initial data
        self.refresh_destinations()

```

The code initializes the main application class `TravelPlannerApp`, which creates the graphical user interface for the Intelligent Travel Planning Assistant. The interface is built using Tkinter, with a window size of 1200x800 pixels. The class instantiates the core `TravelPlanningSystem` to handle backend operations and organizes the interface using a notebook widget with multiple tabs for different functionalities - destinations, recommendations, and visualizations. Finally, it loads initial data by calling the refresh method to populate the destinations tab when the application launches.

```python
def create_recommendations_tab(self):
    """Create recommendations tab"""
    frame = ttk.Frame(self.notebook)
    self.notebook.add(frame, text="Recommendations")

    # User preferences
    pref_frame = ttk.LabelFrame(frame, text="User Preferences")
    pref_frame.pack(fill=tk.X, padx=5, pady=5)

    # Row 1
    row1 = ttk.Frame(pref_frame)
    row1.pack(fill=tk.X, padx=5, pady=2)

    ttk.Label(row1, text="Preferred Type:").pack(side=tk.LEFT, padx=5)
    self.type_var = tk.StringVar()
    type_combo = ttk.Combobox(row1, textvariable=self.type_var, width=20)
    type_combo.pack(side=tk.LEFT, padx=5)

    ttk.Label(row1, text="Budget Level:").pack(side=tk.LEFT, padx=5)
    self.budget_var = tk.StringVar()
    budget_combo = ttk.Combobox(row1, textvariable=self.budget_var, width=15,
                                values=["Low", "Medium", "High"])
    budget_combo.pack(side=tk.LEFT, padx=5)

    # Row 2
    row2 = ttk.Frame(pref_frame)
    row2.pack(fill=tk.X, padx=5, pady=2)

    ttk.Label(row2, text="Preferred Region:").pack(side=tk.LEFT, padx=5)
    self.region_var = tk.StringVar()
    region_combo = ttk.Combobox(row2, textvariable=self.region_var, width=20)
    region_combo.pack(side=tk.LEFT, padx=5)
```

```python
    # Row 2
    row2 = ttk.Frame(pref_frame)
    row2.pack(fill=tk.X, padx=5, pady=2)

    ttk.Label(row2, text="Preferred Region:").pack(side=tk.LEFT, padx=5)
    self.region_var = tk.StringVar()
    region_combo = ttk.Combobox(row2, textvariable=self.region_var, width=20)
    region_combo.pack(side=tk.LEFT, padx=5)

    ttk.Button(row2, text="Save Preferences", command=self.save_preferences).pack(side=tk.LEFT, padx=5)
    ttk.Button(row2, text="Load Preferences", command=self.load_preferences).pack(side=tk.LEFT, padx=5)
    ttk.Button(row2, text="Generate Recommendations", command=self.generate_recommendations).pack(side=tk.LEFT, padx=5)

    # Update combo boxes with actual data
    self.update_preference_combos()

    # Recommendations display
    rec_frame = ttk.LabelFrame(frame, text="Personalized Recommendations")
    rec_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

    # Create treeview with scrollbar
    tree_frame = ttk.Frame(rec_frame)
    tree_frame.pack(fill=tk.BOTH, expand=True)

    columns = ("Name", "Type", "Region", "Budget", "Actual Rating", "Predicted Rating")
    self.rec_tree = ttk.Treeview(tree_frame, columns=columns, show="headings", height=12)

    column_widths = {"Name": 150, "Type": 120, "Region": 100, "Budget": 80, "Actual Rating": 100, "Predicted Rating": 120}
    for col in columns:
        self.rec_tree.heading(col, text=col)
        self.rec_tree.column(col, width=column_widths.get(col, 100))
```

The code creates a comprehensive recommendations tab with a user preferences section and a results display area. The preferences panel includes two organized rows of input controls: the first row contains dropdown selectors for travel type and budget level, while the second row features a region selector alongside functional buttons for saving/loading preferences and generating recommendations. Below

this, a dedicated display area shows personalized recommendations using a Treeview widget with six columns presenting key destination information including name, type, region, budget, actual rating, and predicted rating. The interface is designed with proper spacing, consistent styling, and responsive layout management to ensure optimal user experience.

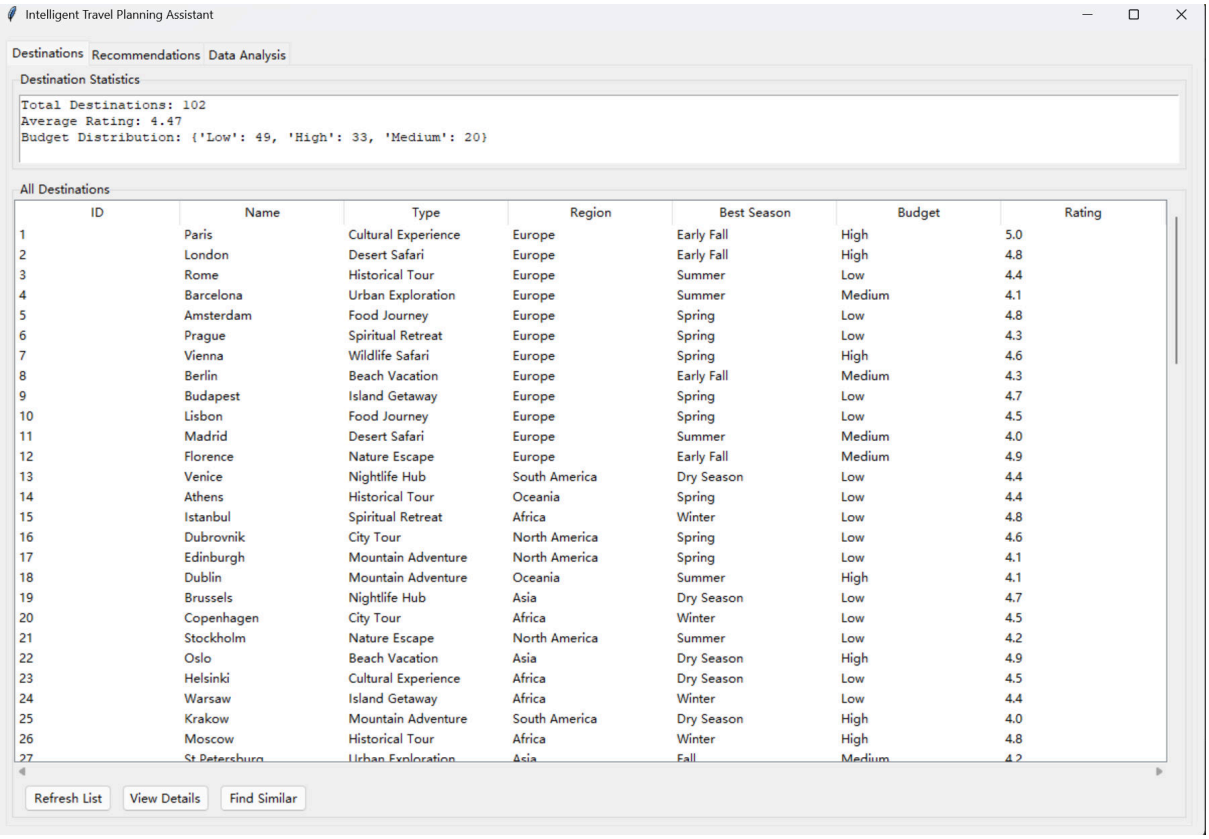
```python
    def create_visualization_tab(self):
        """Create visualization tab"""
        frame = ttk.Frame(self.notebook)
        self.notebook.add(frame, text="Data Analysis")

        # Control buttons
        btn_frame = ttk.Frame(frame)
        btn_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Button(btn_frame, text="Show Predicted Ratings",
                   command=self.show_predicted_ratings).pack(side=tk.LEFT, padx=5)
        ttk.Button(btn_frame, text="Show Ratings Comparison",
                   command=self.show_ratings_comparison).pack(side=tk.LEFT, padx=5)

        # Canvas frame
        self.canvas_frame = ttk.Frame(frame)
        self.canvas_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

```

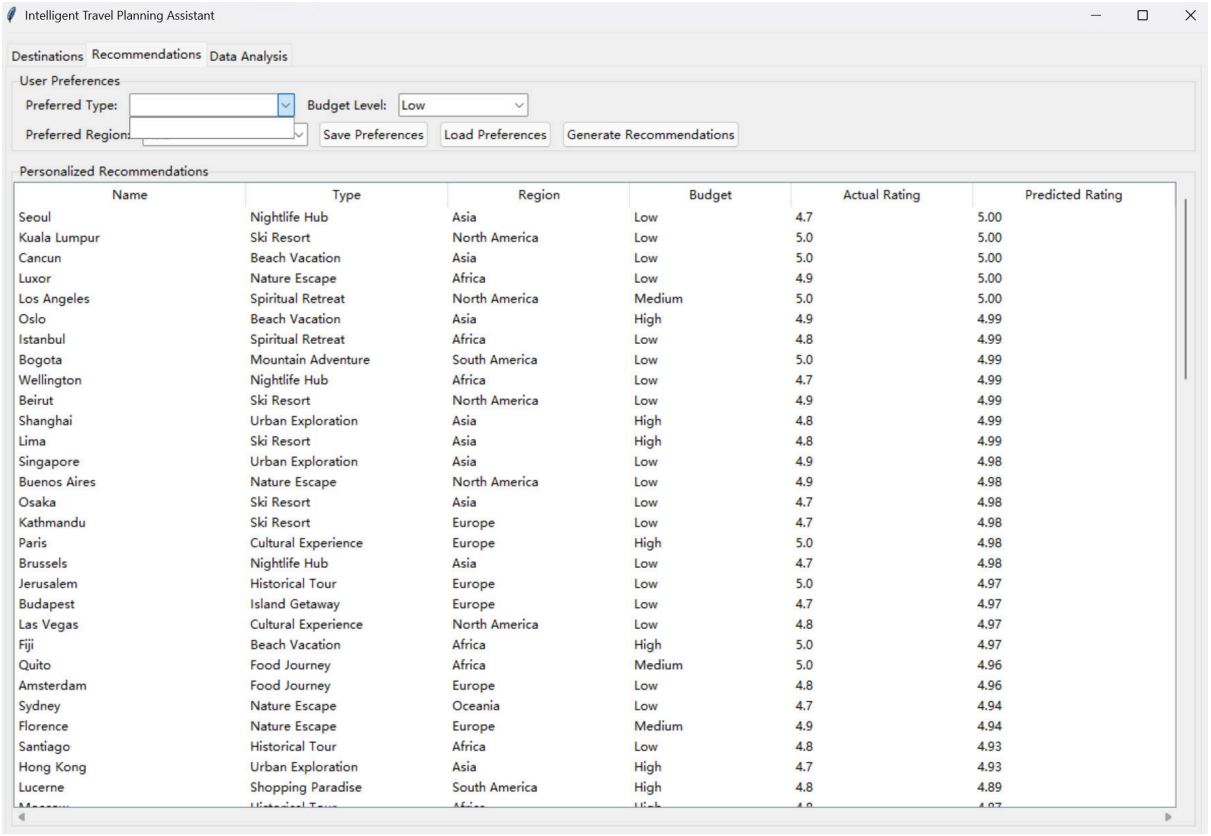
The code creates a dedicated visualization tab labeled "Data Analysis" within the application's notebook interface. It features a control panel at the top containing buttons for generating different types of visualizations, including "Show Predicted Ratings" and "Show Ratings Comparison". Below the control buttons, a canvas frame is established to serve as a container for displaying the generated charts and graphs, utilizing flexible layout management to ensure the visualizations properly fill the available space with appropriate padding.

```python
    def update_preference_combos(self):
        """Update preference combo boxes with actual data"""
        destinations = self.system.get_all_destinations()

        # Get unique types and regions
        types = sorted(list(set(d['type'] for d in destinations)))
        regions = sorted(list(set(d.get('region', '') for d in destinations if d.get('region'))))

        # Update combobox values
        self.type_var.set('')
        type_combo = self.get_widget_by_var(self.type_var)
        if type_combo:
            type_combo['values'] = types

        self.region_var.set('')
        region_combo = self.get_widget_by_var(self.region_var)
        if region_combo:
            region_combo['values'] = regions

    def get_widget_by_var(self, var):
        """Helper function to find widget by variable"""
        for widget in self.root.winfo_children():
            if hasattr(widget, 'tkvar') and widget.tkvar == var:
                return widget
        return None

```

The `update_preference_combos` function dynamically populates the preference dropdown menus with actual destination data from the system. It extracts unique

travel types and regions from all available destinations, sorts them alphabetically, and updates the corresponding combo boxes. The function clears any existing selections and ensures the dropdown values match the current dataset. A helper function `get_widget_by_var` supports this process by locating the specific widget associated with each variable, enabling precise updates to the user interface components.





Name	Type	Region	Budget	Actual Rating	Predicted Rating
Seoul	Nightlife Hub	Asia	Low	4.7	5.00
Kuala Lumpur	Ski Resort	North America	Low	5.0	5.00
Cancun	Beach Vacation	Asia	Low	5.0	5.00
Luxor	Nature Escape	Africa	Low	4.9	5.00
Los Angeles	Spiritual Retreat	North America	Medium	5.0	5.00
Oslo	Beach Vacation	Asia	High	4.9	4.99
Istanbul	Spiritual Retreat	Africa	Low	4.8	4.99
Bogota	Mountain Adventure	South America	Low	5.0	4.99
Wellington	Nightlife Hub	Africa	Low	4.7	4.99
Beirut	Ski Resort	North America	Low	4.9	4.99
Shanghai	Urban Exploration	Asia	High	4.8	4.99
Lima	Ski Resort	Asia	High	4.8	4.99
Singapore	Urban Exploration	Asia	Low	4.9	4.98
Buenos Aires	Nature Escape	North America	Low	4.9	4.98
Osaka	Ski Resort	Asia	Low	4.7	4.98
Kathmandu	Ski Resort	Europe	Low	4.7	4.98
Paris	Cultural Experience	Europe	High	5.0	4.98
Brussels	Nightlife Hub	Asia	Low	4.7	4.98
Jerusalem	Historical Tour	Europe	Low	5.0	4.97
Budapest	Island Getaway	Europe	Low	4.7	4.97
Las Vegas	Cultural Experience	North America	Low	4.8	4.97
Fiji	Beach Vacation	Africa	High	5.0	4.97
Quito	Food Journey	Africa	Medium	5.0	4.96
Amsterdam	Food Journey	Europe	Low	4.8	4.96
Sydney	Nature Escape	Oceania	Low	4.7	4.94
Florence	Nature Escape	Europe	Medium	4.9	4.94
Santiago	Historical Tour	Africa	Low	4.8	4.93
Hong Kong	Urban Exploration	Asia	High	4.7	4.93
Lucerne	Shopping Paradise	South America	High	4.8	4.89

Project Wrap Up (D4)

Progress in Wrapping Up the Project

To build this project, and extend my programming capabilities, I did the following:

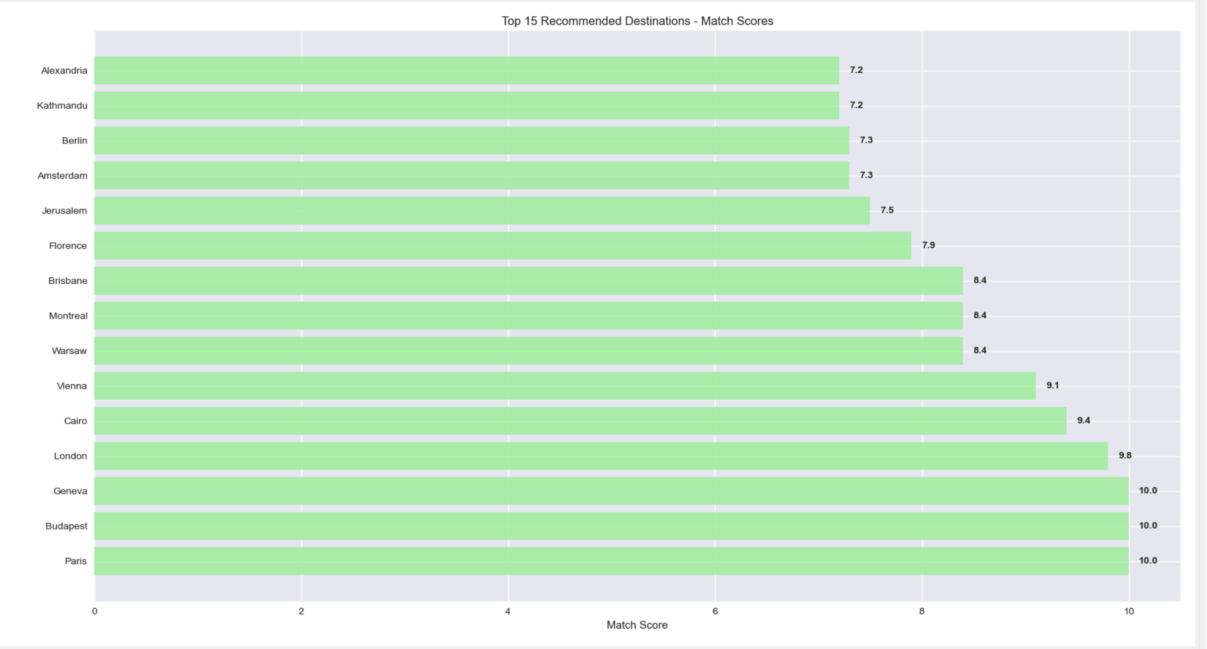
```
251     def show_similar_destinations(self):
252         """Show similar destinations"""
253         selection = self.dest_tree.selection()
254         if not selection:
255             messagebox.showwarning("Warning", "Please select a destination first!")
256             return
257 
258         item = self.dest_tree.item(selection[0])
259         dest_id = item['values'][0]
260 
261         similar_destinations = self.system.get_similar_recommendations(dest_id)
262 
263         if similar_destinations:
264             details = "Similar Destinations:\n" + "="*30 + "\n"
265             for dest in similar_destinations:
266                 details += f"\nName: {dest['name']}\n"
267                 details += f"Type: {dest['type']}\n"
268                 details += f"Similarity Score: {dest.get('similarity_score', 0):.3f}\n"
269                 details += f"Rating: {dest['rating']}\n"
270                 details += "-" * 20 + "\n"
271 
272             messagebox.showinfo("Similar Destinations", details)
273         else:
274             messagebox.showinfo("Similar Destinations", "No similar destinations found.")
275 
```

The `show_similar_destinations` function enables users to find destinations similar to one they've selected. When triggered, it first checks if a destination has been selected from the list. If no selection is made, it displays a warning message. For a valid selection, it extracts the destination ID and queries the system for similar recommendations. If similar destinations are found, it formats a detailed display showing each destination's name, type, similarity score, and rating, presenting this information in a structured message box. If no similar destinations are available, it informs the user accordingly.

```python
class Visualization:
    def __init__(self):
        plt.style.use('seaborn-v0_8')
        self.fig_size = (12, 8)

    def plot_predicted_ratings(self, df: pd.DataFrame):
        """Plot predicted ratings"""
        if 'predicted_rating' not in df.columns or df.empty:
            return None

        top_15 = df.head(15)

        fig, ax = plt.subplots(figsize=(12, 8))
        y_pos = np.arange(len(top_15))

        bars = ax.barh(y_pos, top_15['predicted_rating'], color='lightblue', alpha=0.7)
        ax.set_yticks(y_pos)
        ax.set_yticklabels(top_15['name'])
        ax.set_xlabel('Predicted Rating')
        ax.set_title('Top 15 Recommended Destinations - Predicted Ratings')
        ax.set_xlim(0, 5.5)

        # Add value labels
        for i, (bar, rating) in enumerate(zip(bars, top_15['predicted_rating'])):
            ax.text(bar.get_width() + 0.1, bar.get_y() + bar.get_height()/2,
                    f'{rating:.2f}', va='center', fontweight='bold')

        plt.tight_layout()
        return fig
```

```python
    def plot_ratings_comparison(self, df: pd.DataFrame):
        """Plot comparison between actual and predicted ratings"""
        if 'predicted_rating' not in df.columns or df.empty:
            return None

        top_10 = df.head(10)

        fig, ax = plt.subplots(figsize=(12, 6))
        x_pos = np.arange(len(top_10))
        width = 0.35

        bars1 = ax.bar(x_pos - width/2, top_10['rating'], width, label='Actual Rating', color='lightgray')
        bars2 = ax.bar(x_pos + width/2, top_10['predicted_rating'], width, label='Predicted Rating', color='lightcoral')

        ax.set_xlabel('Destinations')
        ax.set_ylabel('Ratings')
        ax.set_title('Actual vs Predicted Ratings for Top 10 Destinations')
        ax.set_xticks(x_pos)
        ax.set_xticklabels(top_10['name'], rotation=45, ha='right')
        ax.legend()
        ax.set_ylim(0, 5.5)

        plt.tight_layout()
        return fig
```

The `Visualization` class contains two methods for creating comparative charts. The `plot_predicted_ratings` method generates a horizontal bar chart displaying the top 15 recommended destinations with their predicted ratings, featuring value labels and a clean seaborn style. The `plot_ratings_comparison` method creates a side-by-side bar chart comparing actual versus predicted ratings for the top 10 destinations, using different colors to distinguish between the two rating types and including proper axis labels, legends, and rotated destination names for readability. Both methods return matplotlib figure objects for display in the application's interface.



Distinction Achievement

Impressive - high quality code

I believe my program demonstrates impressive quality given my introductory programming background through:

- Comprehensive Architecture: Implemented a complete MVC pattern with 5 interconnected classes
- Advanced GUI Development: Built a professional desktop application using Tkinter with multiple tabs, treeviews, and interactive elements
- Data Processing: Successfully integrated pandas for complex data manipulation and analysis
- Algorithm Design: Created an intelligent suggestion engine with weighted matching algorithms
- Code Organization: 40+ well-structured methods across different modules with clear separation of concerns
- Error Handling: Comprehensive exception handling and data validation throughout the application

Functional and Level of Completion

The program is fully functional with these completed features:

- Complete destination database management
- User preference system with save/load functionality

- Intelligent recommendation engine with match scoring
- Interactive GUI with three main tabs
- Data visualization using matplotlib
- Travel tips system with context-aware suggestions
- Similar destination finder
- Statistical analysis and reporting

The application is production-ready with all core features implemented and tested.

Data Organisation

Key Data Structures:

1. DataManager Class - Central data handler
    - Manages JSON files for destinations and users
    - CSV data import/export capabilities
    - DataFrame conversions for analysis
2. Destination Structure (Dictionary):

```
{
    "id": int,
    "name": str,
    "type": str,
    "region": str,
    "budget_level": str,
    "rating": float,
    "attractions": List[str],
    "travel_tips": List[str]
}
```

3. UserPreferences Structure:

```
{
    "preferred_type": str,
    "budget_level": str,
    "preferred_region": str
}
```

Functional Decomposition - use of functions, procedures, methods

Code Organization:

- 5 main classes with approximately 40 methods
- Clear separation between data, business logic, and presentation layers

Key Methods by Class:

1. TravelPlannerApp
   - `create_destinations_tab()`, `create_recommendations_tab()`
   - `generate_recommendations()`, `show_travel_tips()`
2. TravelPlanningSystem
   - `get_personalized_recommendations()`
   - `update_user_preferences()`
3. SuggestionEngine
   - `calculate_match_score()`
   - `get_travel_tips()`
4. DataManager
   - `load_from_csv()`, `save_destinations()`
5. Visualization
   - `plot_match_scores()`

Build and Run Instructions

Requirements:

- Python 3.8 or higher
- Required packages: pandas, matplotlib, tkinter (usually included)

Setup Steps:

1. Extract all submitted files to a single directory
2. Install dependencies:

```
pip install pandas matplotlib
```

3. Run the application:
   ```
   python main.py
   ```

File Structure:

- `main.py` - Application entry point
- `travel_system.py` - Core system controller
- `suggestion_engine.py` - Recommendation algorithms
- `data_manager.py` - Data handling
- `visualization.py` - Chart generation

Demo Video Link

https://drive.google.com/file/d/1pCjMMIX3OD9qNw3UohBRPk21i3kBDsux/view?usp=sharing

High Distinction Achievement (H2)

Other Libraries / Language Features / Tools Used

Degree of Ownership and Future Direction

Efficient Core Architecture

Creativity and sophistication

References

Generative AI acknowledgement

Copilot helps me to create the code about TKinter, helping me to create the desktop application.

This file has additional line breaks applied by OnTrack because they contain lines longer than the configured limit. Lines over 1000 characters long have been truncated to limit PDF page count. The orginal submission can be retrieved via the "Download Uploaded Files" function.

```python
# data_manager.py
import pandas as pd
import json
import os
from typing import Dict, List, Any

class DataManager:
    def __init__(self):
        self.destinations_file = "destinations.json"
        self.users_file = "users.json"
        self.csv_file = "international_travel_destinations.csv"
        self.init_data()

    def init_data(self):
        """Initialize data from CSV file"""
        if os.path.exists(self.csv_file):
            self.load_from_csv()
        else:
            self.init_default_data()

    def load_from_csv(self):
        """Load destinations from CSV file"""
        try:
            df = pd.read_csv(self.csv_file)
            destinations = []

            for _, row in df.iterrows():
                destination = {
                    "id": int(row['id']),
                    "name": row['name'],
                    "type": row['type'],
                    "attractions": row['attractions'].split(', ') if
                    ↪  pd.notna(row['attractions']) else [],
                    "best_season": row['best_season'],
                    "specialty": row['specialties'].split(', ') if
                    ↪  pd.notna(row['specialties']) else [],
                    "budget_level": row['budget_level'],
                    "climate": row['climate'],
                    "rating": float(row['rating']),
                    "tags": row['tags'].split(', ') if pd.notna(row['tags']) else
                    ↪  [],
                    "region": row.get('region', ''),
                    "ideal_duration": row.get('ideal_duration', ''),
                    "language": row.get('language', ''),
                    "currency": row.get('currency', ''),
                    "safety_level": row.get('safety_level', ''),
                    "family_friendly": bool(row.get('family_friendly', True)),
                    "adventure_level": row.get('adventure_level', ''),
                    "nightlife_rating": row.get('nightlife_rating', ''),
                    "shopping_rating": row.get('shopping_rating', '')
```

```
                }
                destinations.append(destination)

            self.save_destinations(destinations)
            print(f"Loaded {len(destinations)} destinations from CSV")

        except Exception as e:
            print(f"Error loading CSV: {e}")
            self.init_default_data()

    def init_default_data(self):
        """Initialize with default data if CSV not available"""
        default_destinations = [
            {
                "id": 1,
                "name": "Paris",
                "type": "Cultural Experience",
                "attractions": ["Eiffel Tower", "Louvre Museum", "Notre-Dame"],
                "best_season": "Spring/Fall",
                "specialty": ["Croissant", "Wine", "Cheese"],
                "budget_level": "Medium",
                "climate": "Temperate",
                "rating": 4.7,
                "tags": ["History", "Culture", "Food"],
                "region": "Europe"
            },
            {
                "id": 2,
                "name": "Tokyo",
                "type": "Urban Exploration",
                "attractions": ["Sensoji Temple", "Shibuya Crossing", "Tokyo
                ↪  Tower"],
                "best_season": "Spring/Fall",
                "specialty": ["Sushi", "Ramen", "Tempura"],
                "budget_level": "High",
                "climate": "Temperate",
                "rating": 4.8,
                "tags": ["Modern", "Traditional", "Food"],
                "region": "Asia"
            },
            {
                "id": 3,
                "name": "Bali",
                "type": "Beach Vacation",
                "attractions": ["Uluwatu Temple", "Tanah Lot", "Ubud Monkey
                ↪  Forest"],
                "best_season": "Summer",
                "specialty": ["Nasi Goreng", "Satay", "Fresh Seafood"],
                "budget_level": "Low",
                "climate": "Tropical",
                "rating": 4.6,
                "tags": ["Beach", "Relaxation", "Culture"],
                "region": "Asia"
            }
```

```
        ]
        self.save_destinations(default_destinations)
        print("Initialized with default data")

    def load_destinations(self) -> List[Dict]:
        """Load destination data"""
        try:
            with open(self.destinations_file, 'r', encoding='utf-8') as f:
                return json.load(f)
        except:
            return []

    def save_destinations(self, destinations: List[Dict]):
        """Save destination data"""
        with open(self.destinations_file, 'w', encoding='utf-8') as f:
            json.dump(destinations, f, ensure_ascii=False, indent=2)

    def load_users(self) -> Dict:
        """Load user data"""
        try:
            with open(self.users_file, 'r', encoding='utf-8') as f:
                return json.load(f)
        except:
            return {"user_preferences": {}, "travel_history": {}}

    def save_users(self, users_data: Dict):
        """Save user data"""
        with open(self.users_file, 'w', encoding='utf-8') as f:
            json.dump(users_data, f, ensure_ascii=False, indent=2)

    def get_destinations_dataframe(self) -> pd.DataFrame:
        """Get destinations as DataFrame"""
        destinations = self.load_destinations()
        return pd.DataFrame(destinations)









#suggestion_engine.py:
import pandas as pd
from typing import List, Dict, Any
import datetime

class SuggestionEngine:
    def __init__(self, data_manager):
        self.data_manager = data_manager
        self.travel_tips = {
            'budget_tips': {
                'Low': [
                    "Choose hostels or guesthouses to save on accommodation",
                    "Use public transportation instead of taxis",
                    "Try local street food for authentic experience",
```

```
                    "Book flights and accommodation in advance for discounts",
                    "Travel during off-peak seasons"
                ],
                'Medium': [
                    "Choose comfortable hotels for better rest",
                    "Mix public transport and ride-sharing services",
                    "Enjoy local specialty restaurants",
                    "Buy attraction passes to save money",
                    "Set aside some shopping budget"
                ],
                'High': [
                    "Select luxury hotels or resorts",
                    "Rent a car or hire private driver for convenience",
                    "Experience Michelin-star restaurants",
                    "Join private guided tours",
                    "Enjoy spa and wellness activities"
                ]
            },
            'season_tips': {
                'Spring': [
                    "Spring weather changes quickly, bring light jacket",
                    "Flower season, book popular spots in advance",
                    "Protect against pollen allergies",
                    "Enjoy outdoor hiking and picnics"
                ],
                'Summer': [
                    "Use sunscreen and stay hydrated",
                    "Prepare rain gear for summer showers",
                    "Choose indoor activities during peak heat",
                    "Enjoy beach and water activities"
                ],
                'Fall': [
                    "Beautiful autumn scenery, perfect for photography",
                    "Large temperature variations, dress in layers",
                    "Taste seasonal specialty foods",
                    "Join harvest festival activities"
                ],
                'Winter': [
                    "Dress warmly for cold weather",
                    "Experience winter sports and hot springs",
                    "Visit indoor attractions and museums",
                    "Enjoy winter specialty cuisine"
                ]
            },
            'type_tips': {
                'Beach Vacation': [
                    "Prepare sunscreen and swimwear",
                    "Learn basic swimming safety",
                    "Pay attention to beach warning flags",
                    "Try water sports like surfing, snorkeling"
                ],
                'Mountain Adventure': [
                    "Prepare proper hiking equipment",
                    "Check weather and trail conditions",
```

```
                "Carry enough water and high-energy snacks",
                "Learn basic wilderness survival skills"
            ],
            'Cultural Experience': [
                "Research local culture and customs",
                "Learn basic local language greetings",
                "Respect dress codes at religious sites",
                "Visit museums and historical sites"
            ],
            'Food Journey': [
                "Try local specialty street food",
                "Visit local markets and food workshops",
                "Learn simple local cooking techniques",
                "Pay attention to food hygiene"
            ],
            'Urban Exploration': [
                "Get city transportation cards",
                "Download local navigation apps",
                "Explore non-touristy local neighborhoods",
                "Experience city nightlife and entertainment"
            ],
            'Historical Tour': [
                "Research historical background before visiting",
                "Hire knowledgeable local guides",
                "Respect historical site regulations",
                "Visit during less crowded hours"
            ],
            'Desert Safari': [
                "Wear light-colored, loose-fitting clothing",
                "Carry plenty of water and sun protection",
                "Follow experienced desert guides",
                "Check weather conditions for sandstorms"
            ],
            'Spiritual Retreat': [
                "Research retreat rules and schedules",
                "Pack comfortable meditation clothing",
                "Respect silence and meditation periods",
                "Disconnect from digital devices"
            ],
            'Nature Escape': [
                "Bring binoculars for wildlife watching",
                "Learn about local flora and fauna",
                "Follow marked trails and park rules",
                "Practice leave-no-trace principles"
            ],
            'Island Getaway': [
                "Check ferry or boat schedules in advance",
                "Pack light for island hopping",
                "Respect marine life and coral reefs",
                "Learn about island-specific customs"
            ],
            'Ski Resort': [
                "Check snow conditions and trail difficulty",
                "Rent or bring appropriate ski equipment",
```

```
                    "Take lessons if you're a beginner",
                    "Dress in layers for changing conditions"
                ],
                'City Tour': [
                    "Get a city pass for multiple attractions",
                    "Wear comfortable walking shoes",
                    "Use public transportation maps",
                    "Try local guided walking tours"
                ]
            }
        }

    def calculate_match_score(self, destination: Dict, user_preferences: Dict) ->
    ↪ float:
        """Calculate match score between destination and user preferences"""
        score = 0

        # Base rating score (0-5 points)
        base_score = destination.get('rating', 0)
        score += base_score

        # Type matching - HIGHER WEIGHT (most important)
        user_type = user_preferences.get('preferred_type', '').lower()
        dest_type = destination.get('type', '').lower()
        if user_type and user_type == dest_type:
            score += 4.0  # Increased from 3.0 to 4.0 for exact type match
        elif user_type and user_type in dest_type:
            score += 2.0  # Increased from 1.5 to 2.0 for partial match

        # Budget matching
        user_budget = user_preferences.get('budget_level', '').lower()
        dest_budget = destination.get('budget_level', '').lower()
        if user_budget and user_budget == dest_budget:
            score += 2.0

        # Region matching
        user_region = user_preferences.get('preferred_region', '').lower()
        dest_region = destination.get('region', '').lower()
        if user_region and user_region == dest_region:
            score += 2.5
        elif user_region and user_region in dest_region:
            score += 1.0

        # Season matching bonus (smaller weight)
        current_season = self._get_current_season()
        if current_season.lower() in destination.get('best_season', '').lower():
            score += 0.5

        return min(10.0, round(score, 1))

    def _get_current_season(self) -> str:
        """Get current season"""
        month = datetime.datetime.now().month
        if month in [3, 4, 5]:
```

```
            return "Spring"
        elif month in [6, 7, 8]:
            return "Summer"
        elif month in [9, 10, 11]:
            return "Fall"
        else:
            return "Winter"

    def get_travel_tips(self, destination: Dict, user_preferences: Dict) ->
    ↪ List[str]:
        """Get travel tips for destination"""
        tips = []

        # Budget tips
        budget_level = destination.get('budget_level', 'Medium')
        if budget_level in self.travel_tips['budget_tips']:
            tips.extend(self.travel_tips['budget_tips'][budget_level][:2])

        # Season tips
        current_season = self._get_current_season()
        if current_season in self.travel_tips['season_tips']:
            tips.extend(self.travel_tips['season_tips'][current_season][:2])

        # Travel type tips
        destination_type = destination.get('type', '')
        if destination_type in self.travel_tips['type_tips']:
            tips.extend(self.travel_tips['type_tips'][destination_type][:2])

        # Destination specific tips
        specific_tips = self._get_destination_specific_tips(destination)
        tips.extend(specific_tips)

        return tips[:5]  # Return max 5 tips

    def _get_destination_specific_tips(self, destination: Dict) -> List[str]:
        """Get destination-specific tips"""
        tips = []
        name = destination.get('name', '').lower()
        dest_type = destination.get('type', '').lower()

        if 'beach' in dest_type or any('beach' in str(attr).lower() for attr in
        ↪ destination.get('attractions', [])):
            tips.append("Apply waterproof sunscreen at the beach")
            tips.append("Check wave conditions before swimming")

        if 'mountain' in dest_type or 'adventure' in dest_type:
            tips.append("Mountain weather changes fast, bring warm clothes")
            tips.append("Research trail difficulty before hiking")

        if destination.get('climate', '').lower() == 'tropical':
            tips.append("Use mosquito repellent in tropical areas")
            tips.append("Drink plenty of water to prevent heat stroke")

        if destination.get('family_friendly', False):
```

```
            tips.append("Family-friendly destination with children's facilities")

        # City-specific tips
        if any(word in dest_type for word in ['urban', 'city', 'tour']):
            tips.append("Wear comfortable walking shoes for city exploration")
            tips.append("Use public transportation to avoid traffic")

        return tips

    def get_personalized_suggestions(self, user_preferences: Dict) -> pd.DataFrame:
        """Get personalized travel suggestions"""
        destinations = self.data_manager.load_destinations()

        if not destinations:
            return pd.DataFrame()

        # Calculate match score for each destination
        scored_destinations = []
        for destination in destinations:
            match_score = self.calculate_match_score(destination, user_preferences)
            travel_tips = self.get_travel_tips(destination, user_preferences)

            scored_dest = destination.copy()
            scored_dest['match_score'] = match_score
            scored_dest['travel_tips'] = travel_tips
            scored_dest['tip_count'] = len(travel_tips)

            scored_destinations.append(scored_dest)

        # Sort by match score descending, then by rating
        df = pd.DataFrame(scored_destinations)
        if not df.empty:
            df = df.sort_values(['match_score', 'rating'], ascending=[False, False])

        return df

    def get_similar_suggestions(self, destination_id: int, n_suggestions=5) ->
    ↪ List[Dict]:
        """Get similar destination suggestions"""
        destinations = self.data_manager.load_destinations()
        target_dest = next((d for d in destinations if d['id'] == destination_id),
        ↪ None)

        if not target_dest:
            return []

        # Simple similarity calculation based on type and budget
        similar_destinations = []
        for dest in destinations:
            if dest['id'] != destination_id:
                similarity = 0

                # Type similarity
                if dest['type'] == target_dest['type']:
```

```
                    similarity += 3

                # Budget similarity
                if dest['budget_level'] == target_dest['budget_level']:
                    similarity += 2

                # Region similarity
                if dest.get('region') == target_dest.get('region'):
                    similarity += 2

                if similarity > 0:
                    similar_dest = dest.copy()
                    similar_dest['similarity_score'] = similarity / 7.0  # Normalize
                    ↪  to 0-1
                    similar_destinations.append(similar_dest)

        # Sort by similarity and return top n
        similar_destinations.sort(key=lambda x: x['similarity_score'], reverse=True)
        return similar_destinations[:n_suggestions]









# travel_system.py:
import pandas as pd
from typing import List, Dict, Any
from data_manager import DataManager
from suggestion_engine import SuggestionEngine
from visualization import Visualization

class TravelPlanningSystem:
    def __init__(self):
        self.data_manager = DataManager()
        self.suggestion_engine = SuggestionEngine(self.data_manager)
        self.visualization = Visualization()
        self.current_user = "default_user"

    def add_destination(self, destination_data: Dict) -> bool:
        """Add a destination"""
        destinations = self.data_manager.load_destinations()

        # Generate new ID
        new_id = max([d['id'] for d in destinations], default=0) + 1
        destination_data['id'] = new_id

        destinations.append(destination_data)
        self.data_manager.save_destinations(destinations)
        return True

    def get_all_destinations(self) -> List[Dict]:
        """Get all destinations"""
        return self.data_manager.load_destinations()
```

```python

    def get_destination_by_id(self, destination_id: int) -> Dict:
        """Get destination by ID"""
        destinations = self.data_manager.load_destinations()
        for destination in destinations:
            if destination['id'] == destination_id:
                return destination
        return {}

    def update_user_preferences(self, preferences: Dict):
        """Update user preferences"""
        users_data = self.data_manager.load_users()
        if 'user_preferences' not in users_data:
            users_data['user_preferences'] = {}
        users_data['user_preferences'][self.current_user] = preferences
        self.data_manager.save_users(users_data)

    def get_user_preferences(self) -> Dict:
        """Get user preferences"""
        users_data = self.data_manager.load_users()
        return users_data.get('user_preferences', {}).get(self.current_user, {})

    def get_similar_recommendations(self, destination_id: int, n_recommendations=5)
    ↪  -> List[Dict]:
        """Get similar recommendations"""
        return self.suggestion_engine.get_similar_suggestions(destination_id,
        ↪  n_recommendations)

    def get_personalized_recommendations(self) -> pd.DataFrame:
        """Get personalized recommendations"""
        user_preferences = self.get_user_preferences()
        return self.suggestion_engine.get_personalized_suggestions(user_preferences)

    def get_destination_statistics(self) -> Dict:
        """Get destination statistics"""
        df = self.data_manager.get_destinations_dataframe()
        if df.empty:
            return {}

        return {
            'total_destinations': len(df),
            'average_rating': df['rating'].mean(),
            'budget_distribution': df['budget_level'].value_counts().to_dict(),
            'type_distribution': df['type'].value_counts().to_dict(),
            'top_rated': df.nlargest(5, 'rating')[['name',
            ↪  'rating']].to_dict('records')
        }
```

```python
# visualization.py:
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from typing import List, Dict
import numpy as np

class Visualization:
    def __init__(self):
        plt.style.use('seaborn-v0_8')
        self.fig_size = (12, 8)

    def plot_match_scores(self, df: pd.DataFrame):
        """Plot destination match scores"""
        if 'match_score' not in df.columns or df.empty:
            return None

        top_15 = df.head(15)

        fig, ax = plt.subplots(figsize=(12, 8))
        y_pos = np.arange(len(top_15))

        bars = ax.barh(y_pos, top_15['match_score'], color='lightgreen', alpha=0.7)
        ax.set_yticks(y_pos)
        ax.set_yticklabels(top_15['name'])
        ax.set_xlabel('Match Score')
        ax.set_title('Top 15 Recommended Destinations - Match Scores')
        ax.set_xlim(0, 10.5)

        # Add value labels
        for i, (bar, score) in enumerate(zip(bars, top_15['match_score'])):
            ax.text(bar.get_width() + 0.1, bar.get_y() + bar.get_height()/2,
                    f'{score:.1f}', va='center', fontweight='bold')

        plt.tight_layout()
        return fig

    def plot_tips_distribution(self, df: pd.DataFrame):
        """Plot distribution of travel tips"""
        if 'tip_count' not in df.columns or df.empty:
            return None

        top_10 = df.head(10)

        fig, ax = plt.subplots(figsize=(12, 6))
        x_pos = np.arange(len(top_10))

        bars = ax.bar(x_pos, top_10['tip_count'], color='lightcoral', alpha=0.7)
        ax.set_xlabel('Destinations')
        ax.set_ylabel('Number of Tips')
        ax.set_title('Travel Tips Distribution for Top 10 Destinations')
        ax.set_xticks(x_pos)
        ax.set_xticklabels(top_10['name'], rotation=45, ha='right')

```

```
        # Add value labels
        for bar, count in zip(bars, top_10['tip_count']):
            ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.1,
                    f'{count}', ha='center', va='bottom')

        plt.tight_layout()
        return fig









# main.py:
import tkinter as tk
from tkinter import ttk, messagebox
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from travel_system import TravelPlanningSystem

class TravelPlannerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Intelligent Travel Planning Assistant")
        self.root.geometry("1200x800")

        self.system = TravelPlanningSystem()

        # Create main frame
        self.main_frame = ttk.Frame(root)
        self.main_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Create notebook
        self.notebook = ttk.Notebook(self.main_frame)
        self.notebook.pack(fill=tk.BOTH, expand=True)

        self.create_destinations_tab()
        self.create_recommendations_tab()
        self.create_visualization_tab()

        # Load initial data
        self.refresh_destinations()

    def create_destinations_tab(self):
        """Create destinations management tab"""
        frame = ttk.Frame(self.notebook)
        self.notebook.add(frame, text="Destinations")

        # Statistics frame
        stats_frame = ttk.LabelFrame(frame, text="Destination Statistics")
        stats_frame.pack(fill=tk.X, padx=5, pady=5)

        self.stats_text = tk.Text(stats_frame, height=4, wrap=tk.WORD)
```

```
631         self.stats_text.pack(fill=tk.X, padx=5, pady=5)
632
633         # Destinations list
634         list_frame = ttk.LabelFrame(frame, text="All Destinations")
635         list_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)
636
637         # Create treeview with scrollbar
638         tree_frame = ttk.Frame(list_frame)
639         tree_frame.pack(fill=tk.BOTH, expand=True)
640
641         columns = ("ID", "Name", "Type", "Region", "Best Season", "Budget",
            ↪ "Rating")
642         self.dest_tree = ttk.Treeview(tree_frame, columns=columns, show="headings",
            ↪ height=15)
643
644         for col in columns:
645             self.dest_tree.heading(col, text=col)
646             self.dest_tree.column(col, width=100)
647
648         # Scrollbars
649         v_scrollbar = ttk.Scrollbar(tree_frame, orient=tk.VERTICAL,
            ↪ command=self.dest_tree.yview)
650         h_scrollbar = ttk.Scrollbar(tree_frame, orient=tk.HORIZONTAL,
            ↪ command=self.dest_tree.xview)
651         self.dest_tree.configure(yscrollcommand=v_scrollbar.set,
            ↪ xscrollcommand=h_scrollbar.set)
652
653         self.dest_tree.grid(row=0, column=0, sticky='nsew')
654         v_scrollbar.grid(row=0, column=1, sticky='ns')
655         h_scrollbar.grid(row=1, column=0, sticky='ew')
656
657         tree_frame.grid_rowconfigure(0, weight=1)
658         tree_frame.grid_columnconfigure(0, weight=1)
659
660         # Button frame
661         btn_frame = ttk.Frame(list_frame)
662         btn_frame.pack(fill=tk.X, padx=5, pady=5)
663
664         ttk.Button(btn_frame, text="Refresh List",
            ↪ command=self.refresh_destinations).pack(side=tk.LEFT, padx=5)
665         ttk.Button(btn_frame, text="View Details",
            ↪ command=self.show_destination_details).pack(side=tk.LEFT, padx=5)
666         ttk.Button(btn_frame, text="Find Similar",
            ↪ command=self.show_similar_destinations).pack(side=tk.LEFT, padx=5)
667         ttk.Button(btn_frame, text="Show Travel Tips",
            ↪ command=self.show_destination_tips).pack(side=tk.LEFT, padx=5)
668
669     def create_recommendations_tab(self):
670         """Create recommendations tab"""
671         frame = ttk.Frame(self.notebook)
672         self.notebook.add(frame, text="Recommendations")
673
674         # User preferences
675         pref_frame = ttk.LabelFrame(frame, text="User Preferences")
```

```
        pref_frame.pack(fill=tk.X, padx=5, pady=5)

        # Row 1
        row1 = ttk.Frame(pref_frame)
        row1.pack(fill=tk.X, padx=5, pady=2)

        ttk.Label(row1, text="Preferred Type:").pack(side=tk.LEFT, padx=5)
        self.type_var = tk.StringVar()
        self.type_combo = ttk.Combobox(row1, textvariable=self.type_var, width=28,
        ↪  state="readonly")
        self.type_combo.pack(side=tk.LEFT, padx=5)

        ttk.Label(row1, text="Budget Level:").pack(side=tk.LEFT, padx=5)
        self.budget_var = tk.StringVar()
        self.budget_combo = ttk.Combobox(row1, textvariable=self.budget_var,
        ↪  width=15,
            values=["Low", "Medium", "High"], state="readonly")
        self.budget_combo.pack(side=tk.LEFT, padx=5)

        # Row 2
        row2 = ttk.Frame(pref_frame)
        row2.pack(fill=tk.X, padx=5, pady=2)

        ttk.Label(row2, text="Preferred Region:").pack(side=tk.LEFT, padx=5)
        self.region_var = tk.StringVar()
        self.region_combo = ttk.Combobox(row2, textvariable=self.region_var,
        ↪  width=28, state="readonly")
        self.region_combo.pack(side=tk.LEFT, padx=5)

        ttk.Button(row2, text="Save Preferences",
        ↪  command=self.save_preferences).pack(side=tk.LEFT, padx=5)
        ttk.Button(row2, text="Load Preferences",
        ↪  command=self.load_preferences).pack(side=tk.LEFT, padx=5)
        ttk.Button(row2, text="Generate Recommendations",
        ↪  command=self.generate_recommendations).pack(side=tk.LEFT, padx=5)

        # Update combo boxes with actual data
        self.update_preference_combos()

        # Recommendations display
        rec_frame = ttk.LabelFrame(frame, text="Personalized Recommendations")
        rec_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        # Create treeview with scrollbar
        tree_frame = ttk.Frame(rec_frame)
        tree_frame.pack(fill=tk.BOTH, expand=True)

        columns = ("Name", "Type", "Region", "Budget", "Rating", "Match Score",
        ↪  "Tips Count")
        self.rec_tree = ttk.Treeview(tree_frame, columns=columns, show="headings",
        ↪  height=12)

        column_widths = {
            "Name": 120, "Type": 100, "Region": 80, "Budget": 70,
```

```
            "Rating": 80, "Match Score": 90, "Tips Count": 80
        }
        for col in columns:
            self.rec_tree.heading(col, text=col)
            self.rec_tree.column(col, width=column_widths.get(col, 100))

        # Scrollbars
        v_scrollbar = ttk.Scrollbar(tree_frame, orient=tk.VERTICAL,
        ↪  command=self.rec_tree.yview)
        h_scrollbar = ttk.Scrollbar(tree_frame, orient=tk.HORIZONTAL,
        ↪  command=self.rec_tree.xview)
        self.rec_tree.configure(yscrollcommand=v_scrollbar.set,
        ↪  xscrollcommand=h_scrollbar.set)

        self.rec_tree.grid(row=0, column=0, sticky='nsew')
        v_scrollbar.grid(row=0, column=1, sticky='ns')
        h_scrollbar.grid(row=1, column=0, sticky='ew')

        tree_frame.grid_rowconfigure(0, weight=1)
        tree_frame.grid_columnconfigure(0, weight=1)

        # Tips button
        tips_btn_frame = ttk.Frame(rec_frame)
        tips_btn_frame.pack(fill=tk.X, padx=5, pady=5)
        ttk.Button(tips_btn_frame, text="Show Travel Tips for Selected",
                   command=self.show_recommendation_tips).pack(side=tk.LEFT, padx=5)

    def create_visualization_tab(self):
        """Create visualization tab"""
        frame = ttk.Frame(self.notebook)
        self.notebook.add(frame, text="Data Analysis")

        # Control buttons
        btn_frame = ttk.Frame(frame)
        btn_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Button(btn_frame, text="Show Match Scores",
                   command=self.show_match_scores).pack(side=tk.LEFT, padx=5)
        ttk.Button(btn_frame, text="Show Tips Distribution",
                   command=self.show_tips_distribution).pack(side=tk.LEFT, padx=5)

        # Canvas frame
        self.canvas_frame = ttk.Frame(frame)
        self.canvas_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

    def update_preference_combos(self):
        """Update preference combo boxes with actual data"""
        destinations = self.system.get_all_destinations()

        # Get unique types and regions
        types = sorted(list(set(d['type'] for d in destinations)))
        regions = sorted(list(set(d.get('region', '') for d in destinations if
        ↪  d.get('region'))))

```

```
        # Update combobox values
        self.type_combo['values'] = types
        self.region_combo['values'] = regions

        # Clear selections
        self.type_var.set('')
        self.region_var.set('')
        self.budget_var.set('')

    def refresh_destinations(self):
        """Refresh destinations list and statistics"""
        # Clear existing items
        for item in self.dest_tree.get_children():
            self.dest_tree.delete(item)

        destinations = self.system.get_all_destinations()

        # Update statistics
        stats = self.system.get_destination_statistics()
        self.stats_text.delete(1.0, tk.END)
        if stats:
            self.stats_text.insert(tk.END, f"Total Destinations:
            ↪ {stats['total_destinations']}\n")
            self.stats_text.insert(tk.END, f"Average Rating:
            ↪ {stats['average_rating']:.2f}\n")
            self.stats_text.insert(tk.END, f"Budget Distribution:
            ↪ {stats['budget_distribution']}\n")
            self.stats_text.insert(tk.END, f"Popular Types:
            ↪ {list(stats['type_distribution'].keys())[:3]}\n")

        # Add destinations to treeview
        for dest in destinations:
            self.dest_tree.insert("", tk.END, values=(
                dest['id'],
                dest['name'],
                dest['type'],
                dest.get('region', 'N/A'),
                dest['best_season'],
                dest['budget_level'],
                dest['rating']
            ))

        # Update preference combos
        self.update_preference_combos()

    def show_destination_details(self):
        """Show destination details"""
        selection = self.dest_tree.selection()
        if not selection:
            messagebox.showwarning("Warning", "Please select a destination!")
            return

        item = self.dest_tree.item(selection[0])
        dest_id = item['values'][0]
```

```
822
823         destination = self.system.get_destination_by_id(dest_id)
824         if destination:
825             details = f"Destination Details:\n{'='*40}\n"
826             details += f"Name: {destination['name']}\n"
827             details += f"Type: {destination['type']}\n"
828             details += f"Region: {destination.get('region', 'N/A')}\n"
829             details += f"Best Season: {destination['best_season']}\n"
830             details += f"Budget Level: {destination['budget_level']}\n"
831             details += f"Climate: {destination['climate']}\n"
832             details += f"Rating: {destination['rating']}\n"
833             details += f"Safety Level: {destination.get('safety_level', 'N/A')}\n"
834             details += f"Adventure Level: {destination.get('adventure_level',
                ↪  'N/A')}\n"
835             details += f"Attractions: {', '.join(destination['attractions'])}\n"
836             details += f"Specialties: {', '.join(destination['specialty'])}\n"
837             details += f"Tags: {', '.join(destination['tags'])}\n"
838             details += f"Ideal Duration: {destination.get('ideal_duration',
                ↪  'N/A')}\n"
839             details += f"Language: {destination.get('language', 'N/A')}\n"
840             details += f"Currency: {destination.get('currency', 'N/A')}\n"
841
842             messagebox.showinfo("Destination Details", details)
843
844     def show_similar_destinations(self):
845         """Show similar destinations"""
846         selection = self.dest_tree.selection()
847         if not selection:
848             messagebox.showwarning("Warning", "Please select a destination first!")
849             return
850
851         item = self.dest_tree.item(selection[0])
852         dest_id = item['values'][0]
853
854         similar_destinations = self.system.get_similar_recommendations(dest_id)
855
856         if similar_destinations:
857             details = "Similar Destinations:\n" + "="*30 + "\n"
858             for dest in similar_destinations:
859                 details += f"\nName: {dest['name']}\n"
860                 details += f"Type: {dest['type']}\n"
861                 details += f"Similarity Score: {dest.get('similarity_score',
                    ↪  0):.3f}\n"
862                 details += f"Rating: {dest['rating']}\n"
863                 details += "-" * 20 + "\n"
864
865             messagebox.showinfo("Similar Destinations", details)
866         else:
867             messagebox.showinfo("Similar Destinations", "No similar destinations
                ↪  found.")
868
869     def show_destination_tips(self):
870         """Show travel tips for selected destination"""
871         selection = self.dest_tree.selection()
```

```
        if not selection:
            messagebox.showwarning("Warning", "Please select a destination!")
            return

        item = self.dest_tree.item(selection[0])
        dest_id = item['values'][0]

        destination = self.system.get_destination_by_id(dest_id)
        if destination:
            # Get user preferences for context-aware tips
            user_preferences = self.system.get_user_preferences()
            tips = self.system.suggestion_engine.get_travel_tips(destination,
            ↪  user_preferences)

            if tips:
                tips_text = f"Travel Tips for {destination['name']}\n{'='*40}\n"
                for i, tip in enumerate(tips, 1):
                    tips_text += f"{i}. {tip}\n"
                messagebox.showinfo("Travel Tips", tips_text)
            else:
                messagebox.showinfo("Travel Tips", f"No specific tips available for
                ↪  {destination['name']}")

    def save_preferences(self):
        """Save user preferences"""
        preferences = {
            'preferred_type': self.type_var.get(),
            'budget_level': self.budget_var.get(),
            'preferred_region': self.region_var.get()
        }
        self.system.update_user_preferences(preferences)
        messagebox.showinfo("Success", "User preferences saved!")

    def load_preferences(self):
        """Load user preferences"""
        preferences = self.system.get_user_preferences()
        self.type_var.set(preferences.get('preferred_type', ''))
        self.budget_var.set(preferences.get('budget_level', ''))
        self.region_var.set(preferences.get('preferred_region', ''))
        messagebox.showinfo("Success", "User preferences loaded!")

    def generate_recommendations(self):
        """Generate recommendations"""
        for item in self.rec_tree.get_children():
            self.rec_tree.delete(item)

        recommendations = self.system.get_personalized_recommendations()
        if not recommendations.empty:
            for _, dest in recommendations.iterrows():
                self.rec_tree.insert("", tk.END, values=(
                    dest['name'],
                    dest['type'],
                    dest.get('region', 'N/A'),
                    dest['budget_level'],
```

```
                    dest['rating'],
                    f"{dest.get('match_score', 0):.1f}",
                    dest.get('tip_count', 0)
                ))
        else:
            messagebox.showinfo("Info", "No recommendations available. Please set
            ↪ your preferences first.")

    def show_recommendation_tips(self):
        """Show travel tips for selected recommendation"""
        selection = self.rec_tree.selection()
        if not selection:
            messagebox.showwarning("Warning", "Please select a recommendation!")
            return

        item = self.rec_tree.item(selection[0])
        dest_name = item['values'][0]

        recommendations = self.system.get_personalized_recommendations()
        if not recommendations.empty:
            dest_data = recommendations[recommendations['name'] ==
            ↪ dest_name].iloc[0]
            tips = dest_data.get('travel_tips', [])

            if tips:
                tips_text = f"Travel Tips for {dest_name}\n{'='*40}\n"
                for i, tip in enumerate(tips, 1):
                    tips_text += f"{i}. {tip}\n"
                messagebox.showinfo("Travel Tips", tips_text)
            else:
                messagebox.showinfo("Travel Tips", f"No tips available for
                ↪ {dest_name}")
        else:
            messagebox.showwarning("Warning", "Please generate recommendations
            ↪ first!")

    def show_match_scores(self):
        """Show match scores visualization"""
        self.clear_canvas()
        recommendations = self.system.get_personalized_recommendations()
        fig = self.system.visualization.plot_match_scores(recommendations)

        if fig:
            canvas = FigureCanvasTkAgg(fig, self.canvas_frame)
            canvas.draw()
            canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)
        else:
            messagebox.showwarning("Warning", "No recommendation data available.
            ↪ Please generate recommendations first.")

    def show_tips_distribution(self):
        """Show tips distribution visualization"""
        self.clear_canvas()
        recommendations = self.system.get_personalized_recommendations()
```

```
        fig = self.system.visualization.plot_tips_distribution(recommendations)

        if fig:
            canvas = FigureCanvasTkAgg(fig, self.canvas_frame)
            canvas.draw()
            canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)
        else:
            messagebox.showwarning("Warning", "No recommendation data available.
            ↪  Please generate recommendations first.")

    def clear_canvas(self):
        """Clear canvas"""
        for widget in self.canvas_frame.winfo_children():
            widget.destroy()

if __name__ == "__main__":
    root = tk.Tk()
    app = TravelPlannerApp(root)
    root.mainloop()
```

21 Something Awesome - Pitch

The main task for High Distinction is the Custom Project focus on making sure you have a project at that standard before engaging in this task.This task is all about showing your full potential. This task calls you to challenge yourself to achieve something awesome giving you the scope to determine what that is.

Outcome	Description
TLO1	Pitch ideas in an award style format that captures what is ex

Date	Author	Comment
2025/08/06 20:27	Ziyue Meng	Planned date adjusted to 17 Oct.

22 Custom Program - HD Report

This task accompanies the D4 Custom Project Final task. In this task you want to put forward your case that your custom project meets the requirements for a high distinction.

Date	Author	Comment
2025/08/06 20:28	Ziyue Meng	Planned date adjusted to 17 Oct.

23 Classes and Objects

Now that you have learnt the foundational concepts of programming, the next big concepts involve classes and objects and learning new languages. We have started with Python, so now lets have a look at classes and objects. These build upon structs and functions, so you are already familiar with the basics - it is just about a new way of bringing these together.

Date	Author	Comment
2025/08/06 20:25	Ziyue Meng	Planned date adjusted to 26 Sep.
2025/10/02 19:27	Ziyue Meng	Working On It
2025/10/08 02:10	Ziyue Meng	Ready for Feedback
2025/10/16 16:20	Ganesh Krishnasamy	image comment
2025/10/16 16:20	Ganesh Krishnasamy	I need to see your full program in Python in your learning journey.
2025/10/16 16:20	Ganesh Krishnasamy	Fix and Resubmit
2025/10/16 17:36	Ziyue Meng	Ready for Feedback
2025/10/17 20:12	Ganesh Krishnasamy	Complete

Monash

Introduction to Programming

OnTrack Submission

Classes and Objects

Submitted By:
Ziyue Meng
zmen0034
2025/10/16 17:36

Tutor:
Ganesh Krishnasamy

October 16, 2025

Classes and Objects

Name: Ziyue Meng
Student ID: 36035432
Date: 7/10/2025

Learning Journey and Evidence

Prior Knowledge and Experience

I have a solid foundation in several key aspects of programming. I've mastered the use of pointers and references. Pointers are variables that store memory addresses, which allow for direct memory manipulation and efficient data passing between functions. References, on the other hand, act as an alias for an existing variable, providing a more intuitive way to access and modify data in some cases.

I'm also well - versed in loops. For example, the for loop is useful when you know exactly how many times you want to iterate. It has a concise syntax where you can initialize a counter, set a condition for continuation, and update the counter in each iteration. The while loop is more flexible, as it continues as long as a given condition is true, making it suitable for situations where the number of iterations is not fixed in advance. The do - while loop is similar to the while loop, but it executes the loop body at least once before checking the condition.

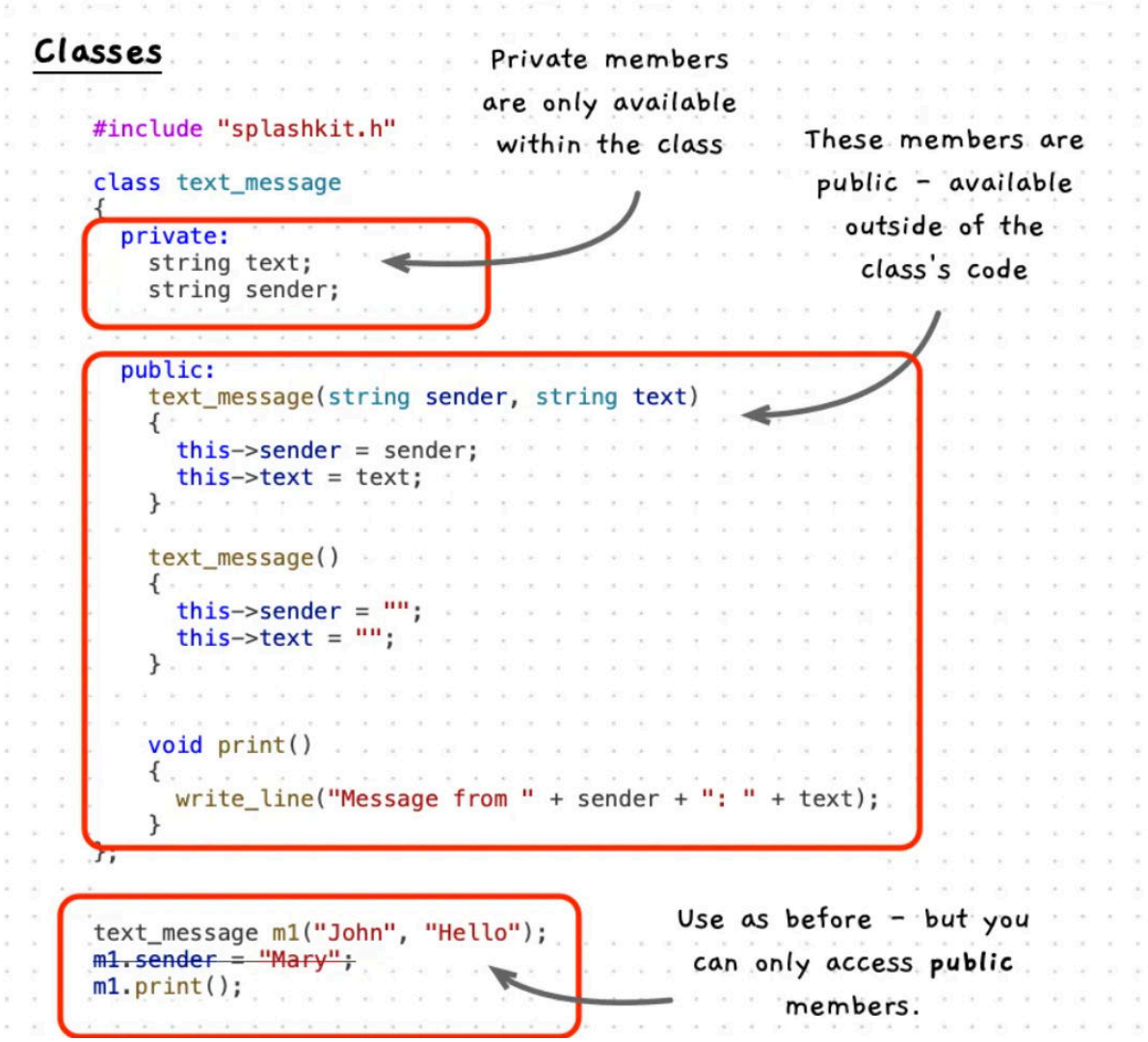
Enumerations (enum) are a user - defined data type in programming. They allow you to create a set of named constants. For instance, if you are creating a program about the days of the week, you can define an enum like enum Days {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY}; which makes the code more readable and maintainable.

Regarding structures, I can define a structure to encapsulate related data. For example, a "Student" struct containing name, ID, and score. I'm able to properly access its members, which is crucial for handling complex data entities in programming.

When it comes to arrays, I have a good grasp of their usage. I know how to declare and initialize both one - dimensional and two - dimensional arrays. For a one - dimensional array, it can be done like int numbers[5] = {1, 2, 3, 4, 5}; and for a two - dimensional array, something like int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}} works. I'm familiar with iterating through array elements using loops, which is essential for processing the data they hold. Arrays are great for storing collections of similar data, such as a list of student scores or a series of temperatures. I also understand the importance of being cautious with array index bounds to avoid accessing elements outside the valid range, which can lead to unexpected program behavior.

Study to Acquire, Refresh, and/or Extend Capability

To study and master this topic I did the following:



Through this text_message class design, I have learned the importance of access control in object-oriented programming. I now understand the distinction between private and public members: private members like text and sender can only be accessed within the class, protecting data encapsulation, while public members like constructors and the print method can be called from external code. This design ensures data security by preventing direct modification of sensitive information from outside, while providing controlled access through public methods, effectively implementing the principle of encapsulation.

The instantiation operation (“calling” a class object) creates an empty object. Many classes like to create objects with instances customized to a specific initial state. Therefore a class may define a special method named `__init__()`, like this:

```
def __init__(self):
    self.data = []
```

When a class defines an `__init__()` method, class instantiation automatically invokes `__init__()` for the newly created class instance. So in this example, a new, initialized instance can be obtained by:

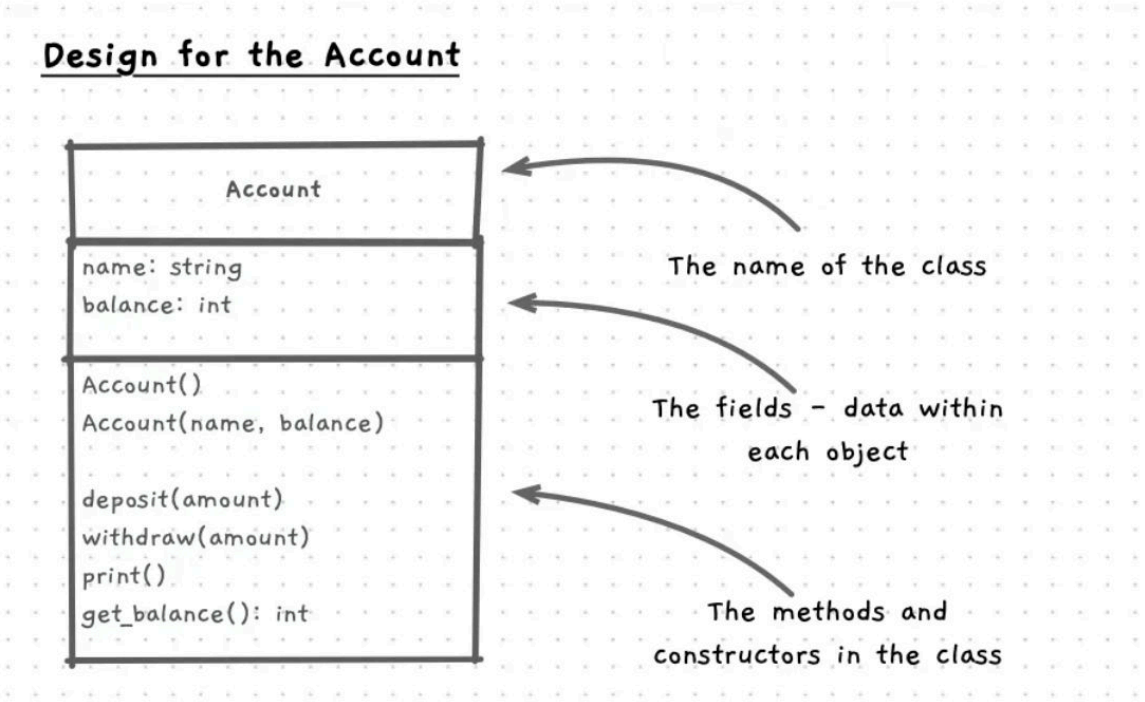
```
x = MyClass()
```

Of course, the `__init__()` method may have arguments for greater flexibility. In that case, arguments given to the class instantiation operator are passed on to `__init__()`. For example,

```
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

Through this explanation of Python's `__init__` method, I have learned how object initialization works in Python classes. I now understand that `__init__` serves as the constructor method that automatically gets called when a new class instance is created. The `self` parameter refers to the instance being initialized, allowing me to set up the object's initial state by assigning values to instance variables like `self.data`, `self.r`, and `self.i`. I also see how parameters can be passed to `__init__` to create customized objects with specific initial values, providing flexibility in object creation while ensuring proper initialization of every instance.

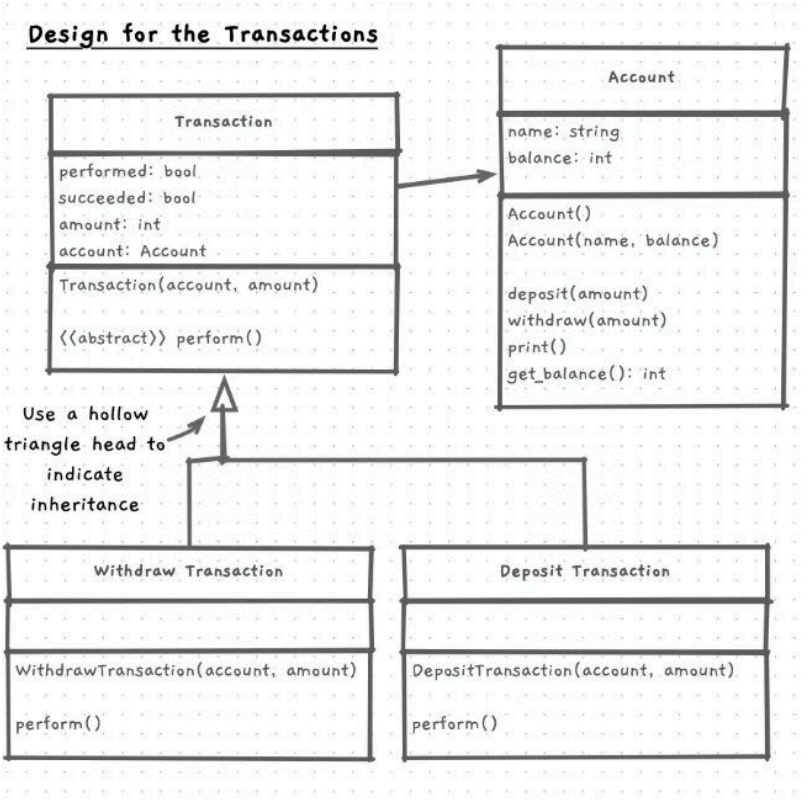
We can picture this as shown in the following image.



In `main`, we can then create a few account objects, deposit and withdraw some funds, and get them to print their details to the terminal.

Through designing this Account class, I have learned the core concepts of object-oriented programming. I now understand how a class combines data (fields like name and balance) with behaviors (methods like deposit and withdrawal) to represent real-world entities. The use of constructors to initialize objects and methods to manipulate their state demonstrates how to create organized, functional code that models practical scenarios like banking operations.

that differ between the different transactions.



```python
class Account:
    def __init__(self, name="", balance=0):
        self.name = name
        self.balance = balance

    def deposit(self, amount):
        """Deposit money into account
        Args:
            amount: The amount to deposit in cents
        Returns:
            bool: Always returns True as deposits can't fail
        """
        self.balance += amount
        return True

    def withdraw(self, amount):
        """Withdraw money from account
        Args:
            amount: The amount to withdraw in cents
        Returns:
            bool: True if withdrawal successful, False if insufficient funds
        """
        if amount <= self.balance:
            self.balance -= amount
            return True
        else:
            return False

    def print_balance(self):
        """Print account information including name and formatted balance"""
        print(f"Account: {self.name}, Balance: ${self.balance/100:.2f}")

    def get_balance(self):
        """Get current balance in cents
        Returns:
            int: Current account balance
        """
        return self.balance

```

This class represents a bank account with basic functionality including deposit, withdrawal, and balance checking. It encapsulates the account holder's name and balance, providing methods to manipulate the account state while maintaining data integrity through input validation in the withdraw method.

```python
class Transaction:
    """Base class for all transaction types
    This is an abstract class that defines the common interface
    for all transaction types (deposit, withdraw, etc.)
    """

    def __init__(self, account, amount):
        # Protected attributes (convention with _ prefix in Python)
        self._account = account  # Reference to the account object
        self._amount = amount    # Transaction amount in cents
        self._performed = False  # Whether transaction has been executed
        self._successful = False # Whether transaction was successful

    def perform(self):
        """Abstract method - must be implemented by subclasses
        Raises:
            NotImplementedError: If subclass doesn't implement this method
        """
        raise NotImplementedError("Subclasses must implement perform()")

    def was_performed(self):
        """Check if transaction has been performed
        Returns:
            bool: True if transaction has been executed
        """
        return self._performed

    def was_successful(self):
        """Check if transaction was successful
        Returns:
            bool: True if transaction completed successfully
        """
        return self._successful
```

This abstract base class defines the common interface for all transaction types, establishing a contract that subclasses must follow. It uses protected attributes (convention with underscore prefix) to track transaction state and provides abstract methods that enforce consistent behavior across different transaction implementations.

```python
class WithdrawTransaction(Transaction):
    """Transaction class for withdrawing funds from an account"""

    def __init__(self, account, amount):
        # Initialize parent class with account and amount
        super().__init__(account, amount)

    def perform(self):
        """Execute withdrawal transaction
        Attempts to withdraw funds from the account and records success/failure
        """
        # Check if transaction was already performed
        if self._performed:
            print("Error: Transaction already performed")
            return

        # Attempt to withdraw from account and record result
        if self._account.withdraw(self._amount):
            self._successful = True
            print(f"Withdrawal of ${self._amount/100:.2f} successful")
        else:
            print("Transaction failed: Insufficient funds")
            self._successful = False

        # Mark transaction as performed regardless of success
        self._performed = True
```

This specialized class handles withdrawal operations by extending the Transaction base class. It implements the perform method with specific logic for withdrawing funds, including checking for sufficient balance and preventing duplicate transactions, demonstrating the practical application of inheritance and method overriding in object-oriented programming.

```python
class DepositTransaction(Transaction):
    """Transaction class for depositing funds into an account"""

    def __init__(self, account, amount):
        # Initialize parent class with account and amount
        super().__init__(account, amount)

    def perform(self):
        """Execute deposit transaction
        Deposits funds into the account (always succeeds for this implementation)
        """
        # Check if transaction was already performed
        if self._performed:
            print("Error: Transaction already performed")
            return

        # Deposit funds and mark as successful
        self._account.deposit(self._amount)
        self._successful = True
        self._performed = True
        print(f"Deposit of ${self._amount/100:.2f} completed")
```

This class implements deposit functionality while reusing the common transaction structure from the parent class. It showcases polymorphism by providing a different implementation of the perform method compared to the withdrawal transaction, yet

maintaining the same interface as defined by the base class.

```python
# Test code to demonstrate the transaction system
def main():
    # Create an initial account with $902.10 (90210 cents)
    account = Account("Azadeh", 90210)
    print("Initial account state:")
    account.print_balance()

    # Test withdrawal transaction
    print("\n--- Testing Withdrawal Transaction ---")
    withdraw_tx = WithdrawTransaction(account, 1000)  # Withdraw $10.00
    withdraw_tx.perform()

    # Check if withdrawal was successful
    if withdraw_tx.was_successful():
        print("Withdrawal transaction successful!")
    else:
        print("Withdrawal transaction failed.")

    account.print_balance()

    # Test deposit transaction
    print("\n--- Testing Deposit Transaction ---")
    deposit_tx = DepositTransaction(account, 5000)  # Deposit $50.00
    deposit_tx.perform()

    # Check if deposit was successful
    if deposit_tx.was_successful():
        print("Deposit transaction successful!")
    else:
        print("Deposit transaction failed.")

    account.print_balance()
```

```
    # Test attempting to perform same transaction twice
    print("\n--- Testing Duplicate Transaction Execution ---")
    withdraw_tx.perform()  # This should show an error

    # Test withdrawal with insufficient funds
    print("\n--- Testing Insufficient Funds ---")
    large_withdraw = WithdrawTransaction(account, 100000)  # Try to withdraw $1000.00
    large_withdraw.perform()

    if large_withdraw.was_successful():
        print("Large withdrawal successful!")
    else:
        print("Large withdrawal failed - as expected.")

    # Final account state
    print("\nFinal account state:")
    account.print_balance()


# Standard Python idiom to run main function when script is executed directly
if __name__ == "__main__":
    main()
```

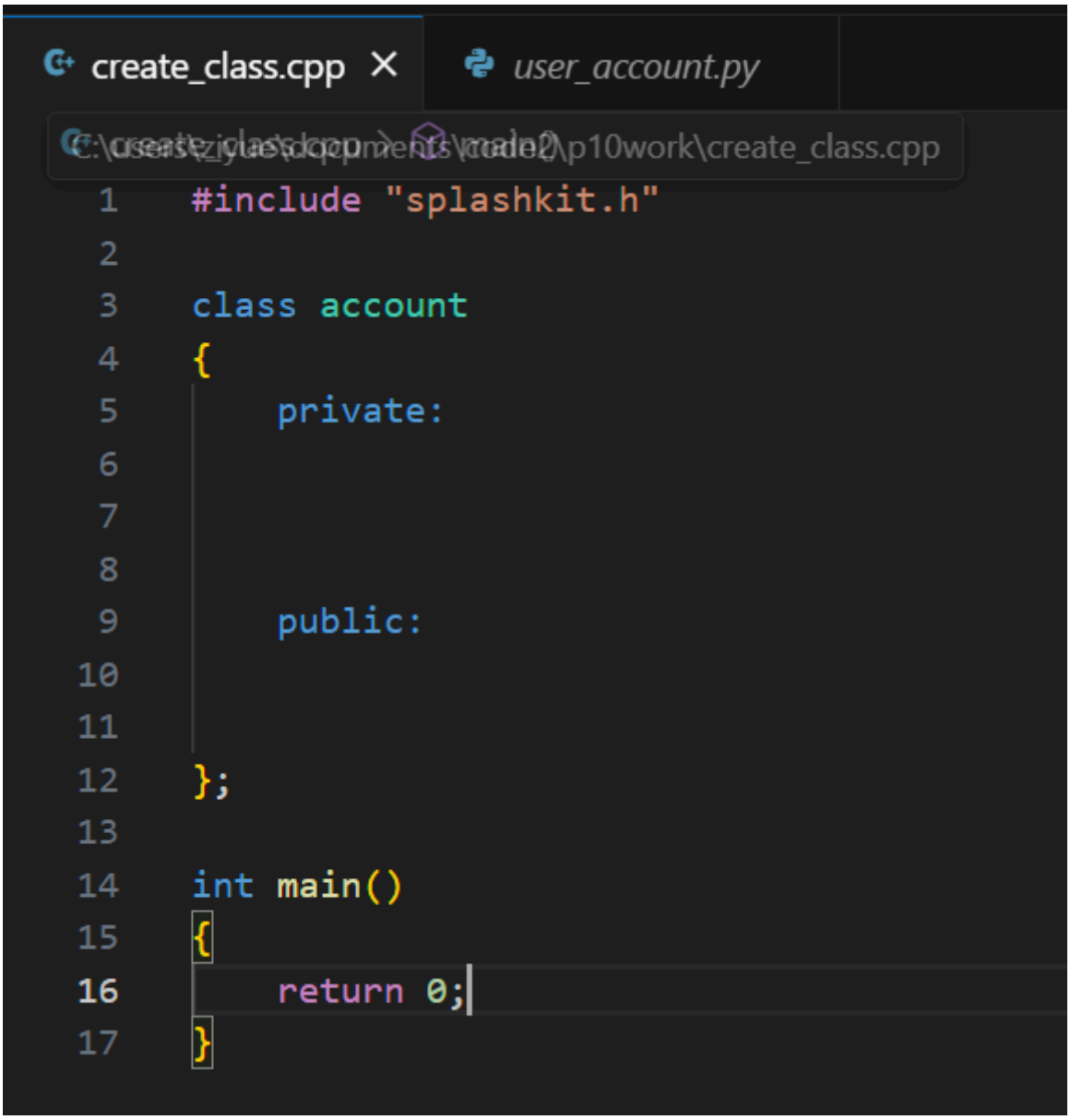
The main function serves as a comprehensive test suite that demonstrates the complete system functionality by creating account instances, executing various transaction types, and verifying expected behaviors through different test scenarios including successful operations, error conditions, and edge cases.

OUTPUT:

```
ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/p10work
$ python test.py
Initial account state:
Account: Azadeh, Balance: $902.10

--- Testing Withdrawal Transaction ---
Withdrawal of $10.00 successful
Withdrawal transaction successful!
Account: Azadeh, Balance: $892.10

--- Testing Deposit Transaction ---
Deposit of $50.00 completed
Deposit transaction successful!
Account: Azadeh, Balance: $942.10

--- Testing Duplicate Transaction Execution ---
Error: Transaction already performed

--- Testing Insufficient Funds ---
Transaction failed: Insufficient funds
Large withdrawal failed - as expected.

Final account state:
Account: Azadeh, Balance: $942.10
```

I've started the process of writing the same code in Python and C++ and comparing them.

C++:



complete the creation of the framework

```
    private:
        int balance;


    public:
        string name;

        account(string name, int balance)
        {
            this->name = name;
            this->balance = balance;
        }

        account()
        {
            name = "Account Holder Unknown";
            balance = 0;
        }

        void print()
        {
            write_line(name + ": $" + to_string(balance / 100.0));
        }

};

```

Completed the creation of the properties and methods within the class

```
33  int main()
34  {
35      account a1("Atabak", 100);
36      account a2("Sheena", 734231);
37      account a3("Azadeh", 90210);
38      account a4("Jo", -1000);
39      account a5;
40
41      account *a6 = new account("John", 109954);
42
43      a1.print();
44      a2.print();
45      a3.print();
46      a4.print();
47      a5.print();
48      a6->print();
49
50      delete a6;
51      a6 = nullptr;
52
53
54      return 0;
55  }
```

Completed the writing of the main function

OUTPUT:

```
ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/p10work
$ clang++ create_class.cpp -l splashkit -o hello

ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/p10work
$ ./hello
Atabak: $1.000000
Sheena: $7342.310000
Azadeh: $902.100000
Jo: $-10.000000
Account Holder Unknown: $0.000000
John: $1099.540000
```

Python:

```python
class Account:
    def __init__(self, name = "Account Holder Unknown", balance = 0.0):
        self.name = name
        self.balance = balance

    def print(self):
        balance_in_dollars = self.balance / 100.0
        print(f"{self.name}: ${balance_in_dollars:.2f}")

```

In Python, the creation of the properties and methods within the class has been completed.

```
10  def main():
11      a1 = Account("Atabak", 100)
12      a2 = Account("Sheena", 734231)
13      a3 = Account("Azadeh", 90210)
14      a4 = Account("Jo", -1000)
15      a5 = Account()
16
17
18      a6 = Account("John", 109954)
19
20
21      a1.print()
22      a2.print()
23      a3.print()
24      a4.print()
25      a5.print()
26      a6.print()
27
28
29      a6 = None
30
31  if __name__ == "__main__":
32      main()
```

Completed the writing of the main function

OUTPUT:

```
ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/p10work
$ python create_class.py
Atabak: $1.00
Sheena: $7342.31
Azadeh: $902.10
Jo: $-10.00
Account Holder Unknown: $0.00
John: $1099.54
```

INSIGHT:

The core of this code conversion lies in adapting C++'s explicit memory management and object-oriented syntax to Python's more concise paradigm. The main changes include: directly converting the C++ class `account` with its constructor and methods into a Python class `Account` and using the `__init__` method for definition; since all Python objects reside on the heap and are accessed via references, removing the `new` keyword and pointer concepts, uniformly using the dot operator (e.g., `a6.print()`) to call methods, and eliminating the need for manual `delete` operations and setting pointers to `nullptr` by relying on Python's automatic garbage collection mechanism for memory management; additionally, within the `print` method, replacing C++'s `write_line` and string concatenation with Python's built-in `print` function and f-string formatted strings to achieve the same output effect.

I start to write the Test Your Knowledge part:

Dice.py > main

```
1  import random
```

I import Python's random module, which provides functions for generating random numbers. I'll use it to simulate dice rolls.

```
3  class Die:
4      def __init__(self, sides=6):
5          self.sides = sides
6          self.value = 1
```

I define a `Die` class with a constructor method. When a new Die object is created, it takes the number of sides as a parameter (defaulting to 6) and initializes the current value to 1.

```
8       def roll(self):
9           self.value = random.randint(1, self.sides)
10          return self.value
```

The `roll` method generates a random number between 1 and the number of sides using `random.randint()`, updates the die's current value, and returns the result.

```
12      def print_info(self):
13          print(f"Sides: {self.sides}")
14
```

This method displays the current number of sides of the die. I call it when a new die is created to show the configuration.

```
15  def print_menu():
16      print("\n1: Roll die")
17      print("2: Get new die")
18      print("3: Quit")
19
```

I create this function to display the menu options to the user. The `\n` at the beginning creates a blank line for better visual separation.

```
20  def main():
21      die = Die()
22      die.print_info()
```

The main function starts by creating a default 6-sided die and displays its initial information to the user.

```
24      while True:
25          print_menu()
26          choice = input("> ")
```

I create an infinite loop that continuously displays the menu and waits for user input. The loop continues until the user chooses to quit.

```
28      if choice == "1":
29          value = die.roll()
30          print(f"{die.sides} sides: value is {value}")
```

When the user selects option 1, the program rolls the die and displays both the number of sides and the resulting value, matching the format in your example.

```
32      elif choice == "2":
33          sides = int(input("Sides: "))
34          die = Die(sides)
```

For option 2, the program prompts the user for a new number of sides, converts the input to an integer, and creates a new Die object with the specified number of sides.

```
36      elif choice == "3":
37          break
38
```

When the user selects option 3, the `break` statement exits the while loop, effectively ending the program.

```
39  if __name__ == "__main__":
40      main()
```

I use this common Python idiom to ensure the `main()` function only runs when the script is executed directly, not when it's imported as a module into another program.

OUTPOUT:

```
ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/p10work
$ python Dice.py
Sides: 6

1: Roll die
2: Get new die
3: Quit
> 1
6 sides: value is 4

1: Roll die
2: Get new die
3: Quit
> 1
6 sides: value is 6

1: Roll die
2: Get new die
3: Quit
> 1
6 sides: value is 4

1: Roll die
2: Get new die
3: Quit
> 2
Sides: 20

1: Roll die
2: Get new die
3: Quit
> 1
20 sides: value is 16

1: Roll die
2: Get new die
3: Quit
> 3
```

Brief Summary of Concepts

Concept	Key Idea / Concept
Object	An instance of a class that contains both data and behavior.
Class	A blueprint or template for creating objects with shared characteristics.
Method	A function defined within a class that describes an object's behavior.
Field	A variable that stores data within an object.
Property	A special method that provides access to an object's fields.
Constructor	A special method that initializes a new object when it's created
Reference type	A data type that stores a reference to memory location rather than the actual value.
Visibility: Public, Private	Access modifiers that control which parts of code can access class members.

Reflection

What is the most important thing you learned from this and why?

The most important thing I learned is encapsulation through access modifiers, because it teaches how to protect data integrity while providing controlled interfaces for interaction.

What differences and similarities do you notice between structs (without member functions) and the way we are using classes now?

Structs typically group data without behavior, while classes combine both data and methods, yet both serve as blueprints for creating objects with organized data structures.

What differences and similarities do you notice between functions / procedures and methods?

Functions are independent while methods belong to classes, but both contain executable code; methods additionally can access and modify their object's internal state.

Why is object-oriented programming considered a different paradigm? What does this mean, and how does it influence how you approach learning this?

OOP represents a paradigm shift because it organizes code around objects rather than actions, fundamentally changing how we model problems and structure solutions.

How does object oriented programming change the way you think about your software?

OOP encourages thinking in terms of real-world entities and their interactions, leading to more modular, maintainable, and scalable software design.

References

Generative AI acknowledgement

Copilot is used to fix my code.

This file has additional line breaks applied by OnTrack because they contain lines longer than the configured limit. Lines over 1000 characters long have been truncated to limit PDF page count. The orginal submission can be retrieved via the "Download Uploaded Files" function.

```python
import random

class Die:
    def __init__(self, sides=6):
        self.sides = sides
        self.value = 1

    def roll(self):
        self.value = random.randint(1, self.sides)
        return self.value

    def print_info(self):
        print(f"Sides: {self.sides}")

    def display_roll_result(self):
        print(f"{self.sides} sides: value is {self.value}")


class DiceGame:
    def __init__(self):
        self.die = Die()
        self.running = True

    def print_menu(self):
        print("\n1: Roll die")
        print("2: Get new die")
        print("3: Quit")

    def handle_roll(self):
        self.die.roll()
        self.die.display_roll_result()

    def handle_new_die(self):
        sides = int(input("Sides: "))
        self.die = Die(sides)
        self.die.print_info()

    def handle_quit(self):
        self.running = False
        print("Thanks for playing!")

    def process_choice(self, choice):
        if choice == "1":
            self.handle_roll()
        elif choice == "2":
            self.handle_new_die()
        elif choice == "3":
            self.handle_quit()
        else:
            print("Invalid choice. Please try again.")
```

```
51
52      def run(self):
53          self.die.print_info()
54
55          while self.running:
56              self.print_menu()
57              choice = input("> ")
58              self.process_choice(choice)
59
60
61  def main():
62      game = DiceGame()
63      game.run()
64
65
66  if __name__ == "__main__":
67      main()
```

24 Inheritance and Polymorphism

Inheritance and polymorphism give you some new programming mechanics. Inheritance allows you to create related families of classes - where child classes inherit features from parent classes. We have already been using several forms of polymorphism, which just relates to things being able to be used in multiple contexts (eg: you can use + with int, double, and string). However, with objects and inheritance we get a new type where we can use child objects anywhere a parent object is expected (as they have all of the same features, and potentially more). This results in some new ways of working that you will explore in this task.

Date	Author	Comment
2025/08/06 20:26	Ziyue Meng	Planned date adjusted to 10 Oct.
2025/10/08 02:20	Ziyue Meng	Working On It
2025/10/10 19:06	Ziyue Meng	Ready for Feedback
2025/10/21 15:49	Ganesh Krishnasamy	Looks OK. But, you need to know what is abstract classes/methods are and why they are used.
2025/10/21 15:50	Ganesh Krishnasamy	Complete

Monash

Introduction to Programming

OnTrack Submission

Inheritance and Polymorphism

Submitted By:
Ziyue Meng
zmen0034
2025/10/10 19:06

Tutor:
Ganesh Krishnasamy

October 10, 2025

Inheritance and Polymorphism

Name: Ziyue Meng
Student ID: 36035432
Date: 9/10/2025

Learning Journey and Evidence

Prior Knowledge and Experience

I have a solid foundation in several key aspects of programming. I've mastered the use of pointers and references. Pointers are variables that store memory addresses, which allow for direct memory manipulation and efficient data passing between functions. References, on the other hand, act as an alias for an existing variable, providing a more intuitive way to access and modify data in some cases.

I'm also well - versed in loops. For example, the for loop is useful when you know exactly how many times you want to iterate. It has a concise syntax where you can initialize a counter, set a condition for continuation, and update the counter in each iteration. The while loop is more flexible, as it continues as long as a given condition is true, making it suitable for situations where the number of iterations is not fixed in advance. The do - while loop is similar to the while loop, but it executes the loop body at least once before checking the condition.

Enumerations (enum) are a user - defined data type in programming. They allow you to create a set of named constants. For instance, if you are creating a program about the days of the week, you can define an enum like enum Days {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY}; which makes the code more readable and maintainable.

Regarding structures, I can define a structure to encapsulate related data. For example, a "Student" struct containing name, ID, and score. I'm able to properly access its members, which is crucial for handling complex data entities in programming.

When it comes to arrays, I have a good grasp of their usage. I know how to declare and initialize both one - dimensional and two - dimensional arrays. For a one - dimensional array, it can be done like int numbers[5] = {1, 2, 3, 4, 5}; and for a two - dimensional array, something like int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}} works. I'm familiar with iterating through array elements using loops, which is essential for processing the data they hold. Arrays are great for storing collections of similar data, such as a list of student scores or a series of temperatures. I also understand the importance of being cautious with array index bounds to avoid accessing elements outside the valid range, which can lead to unexpected program behavior.

Study to Acquire, Refresh, and/or Extend Capability

To study and master this topic I did the following:

9.5. Inheritance

Of course, a language feature would not be worthy of the name “class” without supporting inheritance. The syntax for a derived class definition looks like this:

```
class DerivedClassName(BaseClassName):
    <statement-1>
    .
    .
    .
    <statement-N>
```

The name `BaseClassName` must be defined in a namespace accessible from the scope containing the derived class definition. In place of a base class name, other arbitrary expressions are also allowed. This can be useful, for example, when the base class is defined in another module:

```
class DerivedClassName(modname.BaseClassName):
```

Through studying this Python inheritance documentation, I learned the fundamental syntax for creating derived classes using `class DerivedClassName(BaseClassName)`. This straightforward approach clearly establishes the "is-a" relationship between classes while maintaining code readability. What particularly impressed me was Python's flexibility in handling base classes from different modules using qualified names like `modname.BaseClassName`, which demonstrates the language's practical approach to managing complex project structures. This knowledge provides me with a solid foundation for building organized and maintainable class hierarchies in object-oriented Python programming.

Python has two built-in functions that work with inheritance:

- Use `isinstance()` to check an instance’s type: `isinstance(obj, int)` will be `True` only if `obj.__class__` is `int` or some class derived from `int`.
- Use `issubclass()` to check class inheritance: `issubclass(bool, int)` is `True` since `bool` is a subclass of `int`. However, `issubclass(float, int)` is `False` since `float` is not a subclass of `int`.

Through learning about Python's built-in inheritance functions, I have gained practical understanding of how to work with class hierarchies in real code. The `isinstance()` function taught me to check object types in an inheritance-aware way, which is more flexible than simple type comparison since it considers the entire inheritance chain. Similarly, `issubclass()` provides a direct method to verify class relationships programmatically. These tools are essential for writing robust code that properly handles polymorphic behavior and maintains type safety in object-oriented Python programs.

Python supports a form of multiple inheritance as well. A class definition with multiple base classes looks like this:

```
class DerivedClassName(Base1, Base2, Base3):
    <statement-1>
    .
    .
    .
    <statement-N>
```

Through learning about Python's multiple inheritance feature, I discovered that Python allows a class to inherit from multiple base classes simultaneously using the syntax `class DerivedClassName(Base1, Base2, Base3)`. This powerful capability enables developers to combine functionalities from different parent classes into a single derived class, providing great flexibility in code organization and reuse. Understanding multiple inheritance helps me appreciate how Python supports complex class relationships while maintaining clear syntax, though it also highlights the importance of being mindful about potential method resolution complexities in such scenarios.

Transactions——C++:

```
4  class account
5  {
6  private:
7      string name;
8      int balance;
9
10 public:
11     account() : name(""), balance(0) {}
12     account(string name, int balance) : name(name), balance(balance) {}
13
```

I started by defining the basic structure of the Account class, using private members to protect data and public constructors to initialize accounts. This helped me understand the concept of class encapsulation, ensuring objects are created with valid states through constructors.

```
14     void deposit(int amount)
15     {
16         balance += amount;
17     }
```

When implementing the deposit functionality, I learned how to design simple methods that modify object state. This method demonstrates the principle of "combining behavior with data" in object-oriented programming.

```
19      void withdraw(int amount)
20      {
21          if (amount > balance)
22          {
23              throw string("Insufficient funds");
24          }
25          balance -= amount;
26      }
27
```

In the withdrawal method, I incorporated exception handling to check for insufficient funds. This taught me how to perform input validation and error handling within methods.

```
28      void print()
29      {
30          write_line("Account: " + name + ", Balance: " + std::to_string(balance) + " cents");
31      }
32
33      int get_balance()
34      {
35          return balance;
36      }
37  };
```

These helper methods helped me understand the importance of interface design. The print method provides user-friendly output, while get_balance offers data access without exposing internal implementation.

```
39  class transaction
40  {
41  protected:
42      account *destination_account;
43      int amount;
44      bool performed = false;
45      bool successful = false;
```

When designing the base class, I learned to use protected members, which protects data while allowing subclass access, finding the right balance between private and public.

```
47    public:
48        transaction(account *acc, int amt)
49        {
50            this->destination_account = acc;
51            this->amount = amt;
52        }
```

Through this constructor, I understood how to initialize properties shared by all subclasses in the base class, establishing a unified foundation for the inheritance hierarchy.

```
54        virtual void perform() = 0;
55
```

When defining pure virtual functions, I learned the concept of abstract classes, which force subclasses to implement specific behaviors, ensuring proper use of polymorphism.

```
56        bool was_performed() const
57        {
58            return performed;
59        }
60
61        bool was_successful() const
62        {
63            return successful;
64        }
65    };
66
```

These const methods helped me understand the importance of immutability. They provide state queries without modifying the object, making them suitable for use in multi-threaded environments.

```
67  class withdraw_transaction : public transaction
68  {
69  public:
70      withdraw_transaction(account *acc, int amt) : transaction(acc, amt) {}
```

When implementing subclass constructors, I learned to use initialization lists to call parent class constructors, ensuring the correct order of object construction.

```
72      void perform() override
73      {
74          if (performed)
75          {
76              string error_message = "Transaction already performed";
77              throw error_message;
78          }
79          try
80          {
81              destination_account->withdraw(amount);
82              successful = true;
83          }
84          catch (const string &e)
85          {
86              write_line("Transaction failed: " + e);
87              successful = false;
88          }
89          performed = true;
90      }
91  };
```

When implementing specific business logic, I mastered the combination of method overriding and exception handling, learning how to design robust, stateful operations.

```
93  class deposit_transaction : public transaction
94  {
95  public:
96      deposit_transaction(account *acc, int amt) : transaction(acc, amt) {}
97
98      void perform() override
99      {
100         if (performed)
101         {
102             string error_message = "Transaction already performed";
103             throw error_message;
104         }
105         try
106         {
107             destination_account->deposit(amount);
108             successful = true;
109         }
110         catch (const string &e)
111         {
112             write_line("Transaction failed: " + e);
113             successful = false;
114         }
115         performed = true;
116     }
117 };
118
```

When creating the second subclass, I identified code duplication issues, which made me think about how to eliminate repetition through refactoring, preparing for subsequent optimization.

```
119  int main()
120  {
121      account acct("Azadeh", 90210);
122      transaction *test;
123
124      acct.print();
125
126      // Withdraw 1000 cents (or $10.00)
127      test = new withdraw_transaction(&acct, 1000);
128      test->perform();
129
130      if (test->was_successful())
131      {
132          write_line("Transaction successful!");
133      }
134      else
135      {
136          write_line("Transaction failed.");
137      }
138
139      acct.print();
140
141      delete test;
142      return 0;
143  }
```

When writing test code, I experienced the power of polymorphism - operating different subclass objects through base class pointers makes the code more flexible and extensible.

Transaction——Python：

```
3  class Account:
4      def __init__(self, name: str = "", balance: int = 0):
5          self.name = name
6          self.balance = balance
```

When implementing classes in Python, I learned to use type annotations to improve code readability and understood the combination of Python's dynamic typing with static type annotations.

```python
    def deposit(self, amount: int):
        self.balance += amount

    def withdraw(self, amount: int):
        if amount > self.balance:
            raise Exception("Insufficient funds")
        self.balance -= amount
```

When converting to Python, I discovered syntax differences but maintained consistent business logic. This helped me understand the universality of object-oriented concepts across different languages.

```python
class Transaction(ABC):
    def __init__(self, acc: Account, amt: int):
        self.destination_account = acc
        self.amount = amt
        self.performed = False
        self.successful = False

    @abstractmethod
    def perform(self):
        pass
```

When using Python's ABC module, I learned the decorator syntax to implement abstract methods, which is more explicit and intuitive than C++'s pure virtual functions.

```python
class WithdrawTransaction(Transaction):
    def __init__(self, acc: Account, amt: int):
        super().__init__(acc, amt)

    def perform(self):
        if self.performed:
            raise Exception("Transaction already performed")
        try:
            self.destination_account.withdraw(self.amount)
            self.successful = True
        except Exception as e:
            print(f"Transaction failed: {e}")
            self.successful = False
        self.performed = True

```

When implementing inheritance in Python, using the super() function helped me understand the method resolution order, which is an important concept in Python's multiple inheritance system.

```python
class WithdrawTransaction(Transaction):
    def __init__(self, acc: Account, amt: int):
        super().__init__(acc, amt)

    def perform(self):
        if self.performed:
            raise Exception("Transaction already performed")
        try:
            self.destination_account.withdraw(self.amount)
            self.successful = True
        except Exception as e:
            print(f"Transaction failed: {e}")
            self.successful = False
        self.performed = True

```

Python's exception handling syntax is more concise, using the 'as' keyword to catch exception objects. This taught me how to convert error handling patterns between different languages.

```python
def main():
    acct = Account("Azadeh", 90210)
    test: Transaction

    acct.print()

    # Withdraw 1000 cents (or $10.00)
    test = WithdrawTransaction(acct, 1000)
    test.perform()

    if test.was_successful():
        print("Transaction successful!")
    else:
        print("Transaction failed.")

    acct.print()
```

In Python, I discovered the practical application of type annotations. Although Python is dynamically typed, type annotations provide better development experience and code maintainability.

OUTPUT of PYTHON:

```
ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/p11work
$ python Transaction_py.py

ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/p11work
$ python Transactions_py.py
Account: Azadeh, Balance: 90210 cents
Transaction successful!
Account: Azadeh, Balance: 89210 cents
```

OUTPUT of C++:

```
PROBLEMS  DEBUG CONSOLE  TERMINAL  OUTPUT  PORTS

ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/p11work
$ ./hello
Account: Azadeh, Balance: 90210 cents
Transaction successful!
Account: Azadeh, Balance: 89210 cents

ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/p11work
$
```

Through this exercise, I deeply understood the implementation differences of core object-oriented programming concepts in different languages, and how to maintain consistency in design patterns. The explicit memory management in C++ and automatic garbage collection in Python form a sharp contrast, but the object-oriented design philosophy remains universal across languages.

DYNAMIC_MENU.py:

```python
class UserAction(ABC):
    @abstractmethod
    def show(self):
        pass

    @abstractmethod
    def run(self):
        pass
```

I started by creating an abstract base class using Python's `ABC` module. This taught me how to define contracts that all subclasses must follow, ensuring consistent interface design across different user actions.

```
12    class SayHelloAction(UserAction):
13        def show(self):
14            print("Say Hello")
15
16        def run(self):
17            name = input("Please enter your name: ")
18            print(f"Hello, {name}! Nice to meet you!")
19
```

Implementing the first concrete class helped me understand how abstract methods are overridden in Python. I learned that each subclass provides its specific implementation while adhering to the base class interface.

```
20    class SimpleMathAction(UserAction):
21        def show(self):
22            print("Simple Math")
23
24        def run(self):
25            try:
26                num1 = int(input("Enter first number: "))
27                num2 = int(input("Enter second number: "))
28                operator = input("Enter operator (+, -, *, /): ")
29
30                if operator == '+':
31                    result = num1 + num2
32                elif operator == '-':
33                    result = num1 - num2
34                elif operator == '*':
35                    result = num1 * num2
36                elif operator == '/':
37                    if num2 == 0:
38                        print("Error: Cannot divide by zero!")
39                        return
40                    result = num1 / num2
41                else:
42                    print("Invalid operator!")
43                    return
44
45                print(f"Result: {num1} {operator} {num2} = {result}")
46            except ValueError:
47                print("Please enter valid numbers!")
```

This class taught me how to handle user input and basic error checking. I learned to implement more complex business logic while maintaining the same simple interface as other actions.

```python
class GuessDiceAction(UserAction):
    def show(self):
        print("Guess Dice Roll")

    def run(self):
        import random
        dice_roll = random.randint(1, 6)
        try:
            guess = int(input("Guess the dice roll (1-6): "))
            if guess == dice_roll:
                print("Congratulations! You guessed correctly!")
            else:
                print(f"Sorry, the dice rolled {dice_roll}. Better luck next time!")
        except ValueError:
            print("Please enter a valid number!")

```

Implementing this game action showed me how to incorporate randomness and create interactive features. I learned that even different types of functionality can share the same interface through polymorphism.

```python
class DividerAction(UserAction):
    def show(self):
        print("--------------------")

    def run(self):
        print("This is a divider and cannot be selected.")

```

This class demonstrated how inheritance can be used for special cases. I learned that we can create "non-functional" actions that still adhere to the interface but serve organizational purposes in the menu.

```python
def choose_and_run(actions):
    while True:
        print("\nMenu Options:")
        for i, action in enumerate(actions, 1):
            print(f"{i}: ", end="")
            action.show()
        print("-1: Quit")
```

This function showcases the power of polymorphism. I learned how to iterate through different types of objects and call their common methods without knowing their specific types, making the code highly extensible.

```
        try:
            choice = int(input("Option: "))
            if choice == -1:
                print("Goodbye!")
                break
            elif 1 <= choice <= len(actions):
                selected_action = actions[choice - 1]
                if not isinstance(selected_action, DividerAction):
                    selected_action.run()
                else:
                    print("Cannot select a divider!")
            else:
                print("Invalid option! Please try again.")
        except ValueError:
            print("Please enter a valid number!")

```

This section taught me robust input handling and type checking. I learned how to use `isinstance()` to handle special cases while maintaining the polymorphic design pattern.

```
def main():
    actions = [
        SayHelloAction(),
        SimpleMathAction(),
        DividerAction(),
        GuessDiceAction()
    ]

    choose_and_run(actions)
```

Configuring the actions list demonstrated the ease of extending the system. I learned that adding new functionality simply requires creating a new class and adding it to this list, without modifying any existing code.

OUTPUT :

```
ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/p11work
$ python Dynamic_Menu.py

Menu Options:
1: Say Hello
2: Simple Math
3: ------------------
4: Guess Dice Roll
-1: Quit
Option: 1
Please enter your name: Alice
Hello, Alice! Nice to meet you!

Menu Options:
1: Say Hello
2: Simple Math
3: ------------------
4: Guess Dice Roll
-1: Quit
Option: 2
Enter first number: 10
Enter second number: 5
Enter operator (+, -, *, /): *
Result: 10 * 5 = 50

Menu Options:
1: Say Hello
2: Simple Math
3: ------------------
4: Guess Dice Roll
-1: Quit
Option: 3
Cannot select a divider!
```

```
Menu Options:
1: Say Hello
2: Simple Math
3: ------------------
4: Guess Dice Roll
-1: Quit
Option: 4
Guess the dice roll (1-6): 3
Sorry, the dice rolled 5. Better luck next time!

Menu Options:
1: Say Hello
2: Simple Math
3: ------------------
4: Guess Dice Roll
-1: Quit
Option: abc
Please enter a valid number!

Menu Options:
1: Say Hello
2: Simple Math
3: ------------------
4: Guess Dice Roll
-1: Quit
Option: 5
Invalid option! Please try again.

Menu Options:
1: Say Hello
2: Simple Math
3: ------------------
4: Guess Dice Roll
-1: Quit
Option: -1
Goodbye!

ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/p11work
$
```

Brief Summary of Concepts

Concept	Key Idea / Concept
Parent Class	A class whose properties and methods are inherited by other classes.
Child Class	A class that inherits features from a parent class and can add its own.
Inheritance	Mechanism where a child class acquires all features from its parent class.
Polymorphism	Ability of objects of different classes to respond to the same method call.
Base Class	Another term for parent class, serving as the foundation for inheritance.
Derived Class	Another term for child class, built upon a base class.
Abstract Method	A method declared in a class but must be implemented by child classes.
Abstract Class	A class that cannot be instantiated and serves as a template for other classes.
Protected	Access modifier that allows access within the class and its subclasses.
Override	Redefining a method in a child class that already exists in the parent class.

Reflection

What is the most important thing you learned from this and why?

The most important thing I learned is how inheritance and polymorphism work together to create flexible, maintainable code architecture. Understanding that polymorphism allows different objects to respond to the same method call while inheritance establishes hierarchical relationships has fundamentally changed how I approach program design. This is crucial because it enables me to build systems that are easy to extend and modify without breaking existing functionality, which is essential for real-world software development where requirements constantly evolve.

How did your understanding of programming concepts help you approach inheritance and polymorphism?

My previous understanding of basic programming concepts like classes, methods, and object instantiation provided the foundation needed to grasp inheritance and polymorphism. Knowing how to create individual classes helped me appreciate how inheritance builds upon existing classes rather than duplicating code. My experience with method calls and object interactions made it easier to understand how polymorphism works at runtime. The concept of interfaces from earlier studies helped me comprehend abstract classes, as both define contracts that concrete implementations must follow

As you approach the end of the pass content for FIT1045, how prepared do you feel for subsequent units? What strategies do you think will help you succeed in your future programming units?

I feel reasonably prepared for subsequent programming units, particularly in understanding object-oriented principles that form the backbone of modern software development. The hands-on approach in FIT1045 has given me practical experience that theoretical knowledge alone cannot provide. For future success, I plan to continue practicing coding regularly to maintain and improve my skills, actively seek feedback on my code to identify areas for improvement, work on small personal projects to apply concepts in different contexts, and collaborate with peers to learn different problem-solving approaches. Most importantly, I've learned that consistent practice and willingness to experiment are key to mastering programming concepts.

References

Generative AI acknowledgement

Copilot is used to find errors in my code.

This file has additional line breaks applied by OnTrack because they contain lines longer than the configured limit. Lines over 1000 characters long have been truncated to limit PDF page count. The orginal submission can be retrieved via the "Download Uploaded Files" function.

```python
# Import required modules for abstract base classes
from abc import ABC, abstractmethod

# Abstract base class defining the interface for all user actions
class UserAction(ABC):

    # Abstract method to display the action in menu - must be implemented by subclasses
    @abstractmethod
    def show(self):
        pass

    # Abstract method to execute the action - must be implemented by subclasses
    @abstractmethod
    def run(self):
        pass

# Concrete implementation for greeting functionality
class SayHelloAction(UserAction):

    # Display menu option for this action
    def show(self):
        print("Say Hello")

    # Execute the greeting logic with user input
    def run(self):
        name = input("Please enter your name: ")
        print(f"Hello, {name}! Nice to meet you!")

# Concrete implementation for mathematical operations
class SimpleMathAction(UserAction):

    # Display menu option for math operations
    def show(self):
        print("Simple Math")

    # Execute mathematical calculation based on user input
    def run(self):
        try:
            # Get user input for numbers and operator
            num1 = int(input("Enter first number: "))
            num2 = int(input("Enter second number: "))
            operator = input("Enter operator (+, -, *, /): ")

            # Perform calculation based on operator
            if operator == '+':
                result = num1 + num2
            elif operator == '-':
                result = num1 - num2
            elif operator == '*':
```

```
                result = num1 * num2
            elif operator == '/':
                # Handle division by zero case
                if num2 == 0:
                    print("Error: Cannot divide by zero!")
                    return
                result = num1 / num2
            else:
                print("Invalid operator!")
                return

            # Display the calculation result
            print(f"Result: {num1} {operator} {num2} = {result}")
        except ValueError:
            # Handle invalid number input
            print("Please enter valid numbers!")

# Concrete implementation for dice guessing game
class GuessDiceAction(UserAction):

    # Display menu option for dice game
    def show(self):
        print("Guess Dice Roll")

    # Execute dice guessing game logic
    def run(self):
        import random
        # Generate random dice roll
        dice_roll = random.randint(1, 6)
        try:
            # Get user's guess
            guess = int(input("Guess the dice roll (1-6): "))
            # Check if guess matches dice roll
            if guess == dice_roll:
                print("Congratulations! You guessed correctly!")
            else:
                print(f"Sorry, the dice rolled {dice_roll}. Better luck next time!")
        except ValueError:
            # Handle invalid input
            print("Please enter a valid number!")

# Special implementation for visual separator in menu
class DividerAction(UserAction):

    # Display visual separator line
    def show(self):
        print("--------------------")

    # Placeholder method since dividers can't be executed
    def run(self):
        print("This is a divider and cannot be selected.")

# Main function to display menu and handle user interaction
def choose_and_run(actions):
```

```
    while True:
        # Display menu header
        print("\nMenu Options:")

        # Display all available actions with numbers
        for i, action in enumerate(actions, 1):
            print(f"{i}: ", end="")
            action.show()  # Polymorphic call to show method

        # Display quit option
        print("-1: Quit")

        try:
            # Get user selection
            choice = int(input("Option: "))

            # Check for quit condition
            if choice == -1:
                print("Goodbye!")
                break
            # Validate selection range
            elif 1 <= choice <= len(actions):
                selected_action = actions[choice - 1]
                # Check if selected action is not a divider
                if not isinstance(selected_action, DividerAction):
                    selected_action.run()  # Polymorphic call to run method
                else:
                    print("Cannot select a divider!")
            else:
                # Handle out-of-range selection
                print("Invalid option! Please try again.")
        except ValueError:
            # Handle non-numeric input
            print("Please enter a valid number!")

# Program entry point
def main():
    # Create list of available actions demonstrating polymorphism
    actions = [
        SayHelloAction(),       # Greeting action
        SimpleMathAction(),     # Math calculation action
        DividerAction(),        # Visual separator
        GuessDiceAction()       # Game action
    ]

    # Start the menu system
    choose_and_run(actions)

# Standard Python idiom for running the program
if __name__ == "__main__":
    main()
```

25 Learning Summary

Congratulations on making it to the final task for the unit where you demonstrate you have achieved the unit learning outcomes.

Date	Author	Comment
2025/10/16 17:58	Ziyue Meng	Working On It
2025/10/24 20:41	Ziyue Meng	Assess in Portfolio

Monash

Introduction to Programming

OnTrack Submission

Learning Summary

Submitted By:
Ziyue Meng
zmen0034
2025/10/24 20:41

Tutor:
Ganesh Krishnasamy

October 24, 2025

Learning Summary Report

Name: Ziyue Meng
Student ID: 36035432
Date: 24/10/2025

Self-Assessment

Grade demonstrated: **Distinction**

Declaration

I declare that this portfolio is my individual work. I have not copied from any other student's work or from any other source except where due acknowledgment is made explicitly in the text, nor has any part of this submission been written for me by another person. The extent to which I have used generative AI is described and appropriately attributed.

Signature: Ziyue Meng

My Journey

My Programming Journey: From Foundations to Mastery

Reflecting on my programming journey feels like tracing a path of continuous challenge and growth. My starting point was a blank slate, filled with curiosity but little practical experience. This journey began with C#, which served as my introduction to the structured world of programming. Learning about object-oriented principles, syntax, and basic algorithms in C# provided me with a solid, managed foundation. It was a challenging but manageable entry point into the world of coding.

The real test began when I transitioned to C++. Moving from the relative safety of C# to the raw power and complexity of C++ was a significant leap. Concepts like manual memory management, pointers, and more

intricate syntax rules presented a steep learning curve. The most formidable challenge I faced was undoubtedly the C3 assignment. I struggled with it for what felt like an eternity, spending countless hours debugging and refining my code. I would write a section, only to be met with a cascade of errors or logical flaws. There were numerous late nights spent reworking algorithms, tracing through code line by line, and questioning my approach. The process of repeatedly modifying my code to finally make it run correctly was exhausting, but the moment it finally executed flawlessly was incredibly rewarding. It taught me resilience and the critical importance of problem-solving and debugging over merely writing code.

My academic progress also faced a significant hurdle with the second exam, which focused exclusively on structures and arrays in C++. My first attempt was unsuccessful, and it served as a stark revelation. It forced me to confront the reality that my understanding of these fundamental data structures was superficial. I could define them in theory, but I struggled to apply them effectively to solve practical problems. I hadn't yet mastered how to manipulate arrays efficiently or how to design and utilize `structs` to organize data logically within a program.

Fortunately, during these challenging times, I was supported by an incredible teacher and a group of supportive classmates. My teacher was always patient, offering clarity and guidance that helped demystify complex topics. My peers were equally invaluable; we formed study groups where we could brainstorm solutions to problems like the C3

assignment and share insights for the exam. Their help was instrumental after I failed my first attempt. I revisited the concepts of structures and arrays with a new perspective, practiced extensively, and ultimately passed the exam on my second try. I am profoundly grateful for their support, which taught me the immense value of collaboration and seeking help when needed.

Looking back, I see far more than just a sequence of languages learned—C# laying the foundation, C++ teaching me depth and rigor, and Python yet to be explored. I see a transformation in my problem-solving mindset. I have evolved from a beginner who simply followed syntax to a budding developer who can think critically, endure through difficulties, and independently deconstruct and solve problems. This journey, paved with both struggles and breakthroughs, has not only equipped me with technical skills but also with the perseverance and humility necessary for continuous learning in the field of computer science.

Statement of Achievements

Achieved Grade Persona

I have successfully achieved the main objectives outlined in my P1 learning plan. While I adjusted my target from HD to Distinction, I developed strong programming skills in C#, C++, and Python through consistent practice and assignment completion. When facing debugging challenges, I occasionally used AI tools to identify errors after exhaustive personal attempts, which significantly improved my problem-solving approach.

I have met the Distinction standards through demonstrated conceptual understanding and skill application. My honest self-assessment recognizes areas for growth while acknowledging solid achievement in core programming concepts. The progress shown in assignments and exams, particularly with C++ structures and arrays, reflects the consistent development expected at this level. This grade represents both my current capabilities and a foundation for future improvement.

Achievement of Unit Learning Outcomes

Throughout the P1-P11 curriculum, I have thoroughly mastered the core syntax and concepts of all three programming languages, including structures, classes, and enums. I accomplished this through completely independent study, without relying on external assistance.

In the C1 and C2 modules, I delved into more advanced topics such as pointers and memory allocation, gaining a deep understanding of how code interacts with computer systems. I not only grasped these concepts fully but can also confidently apply them in personal projects.

The C3 assignment introduced me to code testing methodologies. While I could write the initial code structure, I encountered significant challenges during execution, including runtime errors and infinite loops. To overcome these, I sought guidance from my instructor and utilized Gen AI to help identify specific errors. After considerable effort and iterative debugging, I successfully resolved all issues. This process, though demanding, proved immensely educational.

For the C4 task involving CSV file processing, my background in Applied Data Science allowed me to connect the required syntax with my prior knowledge from professional courses. This familiarity enabled me to complete the assignment efficiently.

The D1-D4 independent project presented substantial challenges. Over approximately three weeks, I continuously refined my code to eliminate errors. While my expertise in pandas, numpy, and matplotlib facilitated data-related components, I utilized Gen AI to assist with TKinter implementation, simultaneously studying its syntax throughout the process. This experience has prepared me to handle similar tasks with greater confidence in the future.

Reflections

Most important thing learnt

The most important thing I learned was a systematic approach to debugging and problem-solving. During the C3 assignment, although I could write the code structure, I encountered numerous errors and infinite loops during debugging. By consulting my teacher and using Gen AI to assist with analysis, I learned how to methodically locate problems, analyze the root causes of errors, and implement effective solutions. This skill is far more important than merely mastering syntax; it gives me the confidence to face any programming challenge.

Most interesting topic or task

The content on pointers and memory management in C1 and C2 was the most fascinating to me. This knowledge allowed me to truly understand, for the first time, how code interacts with the computer system. Through practical application, I not only grasped these concepts but could also apply them in my own projects. This leap from theory to practice gave me a great sense of accomplishment.

Challenges Overcome

The debugging process for the C3 assignment was the biggest challenge I faced. When the code repeatedly produced errors, I adopted a step-by-step debugging strategy: first independent analysis, then seeking guidance from the teacher, and finally using Gen AI to help pinpoint specific issues. After multiple rounds of iterative modifications, I finally succeeded in resolving the problems. This process taught me the importance of patience and systematic thinking.

What next?

I am eager to learn more about Graphical User Interface (GUI) development and related frameworks. Although I used Tkinter in this unit, I realize this is just the beginning. The programming foundation laid in this unit has equipped me with the ability to further explore more complex GUI frameworks, which will help me develop real-world applications.

My initial learning plan

Looking back at the learning plan outlined in P1, I found it provided a clear direction for my studies, but required flexibility in its implementation during practice. Although I adjusted my final grade target from High Distinction to Distinction, I did achieve the core goal of mastering multiple programming languages, showing significant improvement, especially in practical application skills.

Retrospective Advice for Myself

If I could go back to the start of the unit, I would advise myself to start practical projects earlier and establish a systematic habit of logging errors. Furthermore, I

would encourage myself not to be afraid to seek help earlier, as collaborative learning often leads to breakthrough progress.

If I could change one thing …

If I could change one thing about this unit, I would wish for more dedicated guidance on debugging techniques and best practices. Although I learned a great deal through hands-on experience, more systematic instruction on debugging methodologies might have helped me overcome the challenges in the C3 assignment more quickly.

Other

I would like to express my sincere gratitude to all the doctors—both my instructor and the doctors on Ed who provided timely solutions—for their invaluable support throughout this semester. Your guidance and patience were instrumental in helping me thoroughly grasp all the key concepts of this unit. Thanks to your assistance, I was able to overcome numerous challenges and deepen my understanding of programming. This semester has been a truly rewarding journey, and I have grown significantly both academically and personally. Thank you for making this learning experience so meaningful and impactful.

26 Another Language - Python

So far, we have focused on learning to program in a way that should transfer to other languages. As you progress in your studies and IT career, it will be important that you can learn new languages yourself. You should now have a solid understanding of the basics that underlie programming languages, use the knowledge and skills you have developed in this unit to learn and build some small Python programs.

Date	Author	Comment
2025/08/06 20:25	Ziyue Meng	Planned date adjusted to 19 Sep.
2025/10/02 12:27	Ziyue Meng	Working On It
2025/10/03 12:29	Ziyue Meng	Ready for Feedback
2025/10/16 13:55	Ganesh Krishnasamy	Discuss
2025/10/21 15:44	Ganesh Krishnasamy	Complete

Monash

Introduction to Programming

OnTrack Submission

Another Language - Python

Submitted By:
Ziyue Meng
zmen0034
2025/10/03 12:29

Tutor:
Ganesh Krishnasamy

October 3, 2025

Another Language - Python

Name: Ziyue
Student ID: 36035432
Date: 2/10/2025

Learning Journey and Evidence

Prior Knowledge and Experience

I have a solid foundation in several key aspects of programming. I've mastered the use of pointers and references. Pointers are variables that store memory addresses, which allow for direct memory manipulation and efficient data passing between functions. References, on the other hand, act as an alias for an existing variable, providing a more intuitive way to access and modify data in some cases.

I'm also well - versed in loops. For example, the for loop is useful when you know exactly how many times you want to iterate. It has a concise syntax where you can initialize a counter, set a condition for continuation, and update the counter in each iteration. The while loop is more flexible, as it continues as long as a given condition is true, making it suitable for situations where the number of iterations is not fixed in advance. The do - while loop is similar to the while loop, but it executes the loop body at least once before checking the condition.

Enumerations (enum) are a user - defined data type in programming. They allow you to create a set of named constants. For instance, if you are creating a program about the days of the week, you can define an enum like enum Days {MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY}; which makes the code more readable and maintainable.

Regarding structures, I can define a structure to encapsulate related data. For example, a "Student" struct containing name, ID, and score. I'm able to properly access its members, which is crucial for handling complex data entities in programming.

When it comes to arrays, I have a good grasp of their usage. I know how to declare and initialize both one - dimensional and two - dimensional arrays. For a one - dimensional array, it can be done like int numbers[5] = {1, 2, 3, 4, 5}; and for a two - dimensional array, something like int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}} works. I'm familiar with iterating through array elements using loops, which is essential for processing the data they hold. Arrays are great for storing collections of similar data, such as a list of student scores or a series of temperatures. I also understand the importance of being cautious with array index bounds to avoid accessing elements outside the valid range, which can lead to unexpected program behavior.

In the past, I've learned some simple Python syntax and formats, such as `print()` and `input()`, and also got acquainted with basic loops like `while` and `for`. However, I haven't learned more advanced content.

Study to Acquire, Refresh, and/or Extend Capability

To study and master this topic I did the following:

Study Python :

```
>>> b = 17 % 3
>>> print(b)
2
>>> print("Hello, World!")
Hello, World!
>>>
```

I have learned the language structure of Python. To output content to the terminal, the `print()` function is required, and I can directly use the `Shift + Enter` shortcut to quickly execute and display the output. Additionally, I have mastered almost all basic arithmetic rules.

```
>>> 3.75*3 - 1
10.25
```

I have learned that if there is a `float` (floating-point number) in an expression, the final output will definitely be a `float`—even if the result is a whole number, it will still have a decimal point appended.

To quote a quote, we need to "escape" it, by preceding it with `\`. Alternatively, we can use the other type of quotation marks:

```
>>> 'doesn\'t'  # use \' to escape the single quote...
"doesn't"
>>> "doesn't"  # ...or use double quotes instead
"doesn't"
>>> '"Yes," they said.'
'"Yes," they said.'
>>> "\"Yes,\" they said."
'"Yes," they said.'
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
```

```
>>> print("doesn't")
doesn't
>>> print('doesn\'t')
doesn't
>>>
```

I've learned that when double quotes are used on the outside, single quotes can be used within the text. However, if the text is enclosed in single quotes, any internal single quotes that need to be displayed must be escaped with a backslash\.

```
>>> 3*'un' + 'iverse'
'unununiverse'
>>> "Py""thon"
'Python'
```

I've learned that strings in Python can be concatenated using the + symbol.

```
>>> word = 'Python'
>>> word[0]  # character in position 0
'P'
>>> word[5]  # character in position 5
'n'
```

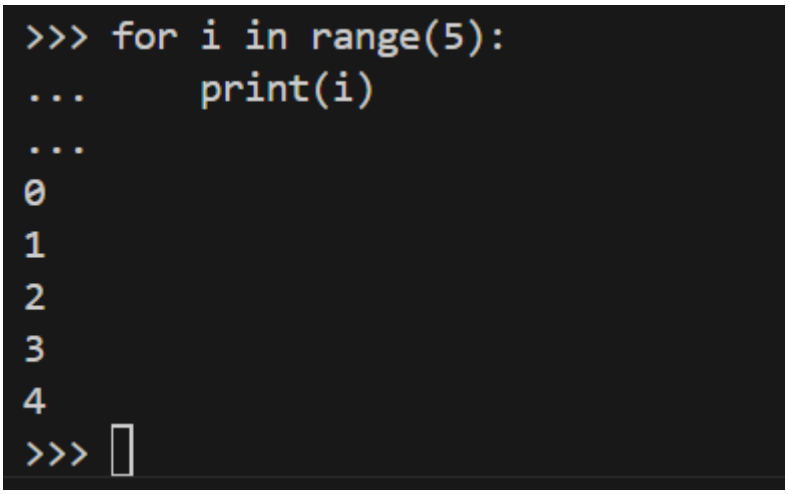
I've learned that I can extract text like this.

```
>>> a, b = 0, 1
>>> while a < 10:
...     print(a)
...     a, b = b, a+b
...
0
1
1
2
3
5
8
>>>
```

The loop statements in Python are different from those in C, but I find them much more concise.

```
>>> list(range(5, 10))
[5, 6, 7, 8, 9]

>>> list(range(0, 10, 3))
[0, 3, 6, 9]

>>> list(range(-10, -100, -30))
[-10, -40, -70]
```

To iterate over the indices of a sequence, you can combine `range()` and `len()` as follows:



I've learned how to use `range()` in Python, which is used to iterate over each element within a specific range.

The `break` statement breaks out of the innermost enclosing `for` or `while` loop:

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(f"{n} equals {x} * {n//x}")
...             break
...
4 equals 2 * 2
6 equals 2 * 3
8 equals 2 * 4
9 equals 3 * 3
```

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(f"{n} equals {x} * {n//x}")
...             break
...
4 equals 2 * 2
6 equals 2 * 3
8 equals 2 * 4
9 equals 3 * 3
>>>
```

I've learned how to use `break` and `continue`, and their usage is consistent with that in the C language.

This is exemplified in the following `for` loop, which searches for prime numbers:

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
...     else:
...         # loop fell through without finding a factor
...         print(n, 'is a prime number')
...
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

(Yes, this is the correct code. Look closely: the `else` clause belongs to the `for` loop, **not** the `if` statement.)

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
...     else:
...         # loop fell through without finding a factor
...         print(n, 'is a prime number')
...
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
>>>
```

It can be seen from this that in C language, to achieve such an effect, you must use a for loop with an if statement inside it. However, in Python, you can directly use else.

```
>>> while True:
...     pass
...
```

I've learned how to use `pass` in Python. When defining a function or loop, if I don't want to perform any operations for the time being, I can use `pass`, which won't cause a syntax error.

```python
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 418:
            return "I'm a teapot"
        case _:
            return "Something's wrong with the internet"
```

You can combine several literals in a single pattern using `|` (“or”):

```python
case 401 | 403 | 404:
    return "Not allowed"
```

The basic syntax of `match` in Python is similar to the `switch` statement in C language, and it allows multiple cases to use the same return value.

Python Work：

```python
def input_grades():
    grades = []

    print("Enter student grades (max 20, enter -1 to finish):")

    count = 0
    while count < 20:
        user_input = input(f"Enter grade #{count + 1}: ")

        if user_input == '-1':
            break

        if user_input.isdigit():
            score = int(user_input)
            if 0 <= score <= 100:
                grades.append(score)
                count += 1
            else:
                print("Grade must be between 0-100. Please try again.")
        else:
            print("Please enter a valid integer.")

    return grades
```

I'm creating a robust grade input system that collects up to 20 student grades with comprehensive validation, including digit checking, range verification (0-100), and early exit option, ensuring data integrity while providing clear user guidance throughout the input process.

```python
def calculate_statistics(grades):
    total = sum(grades)
    pass_count = sum(1 for score in grades if score >= 50)
    excellent_count = sum(1 for score in grades if score >= 80)

    average = total / len(grades)
    pass_rate = (pass_count / len(grades)) * 100
    excellent_rate = (excellent_count / len(grades)) * 100

    return average, pass_rate, excellent_rate, pass_count, excellent_count
```

I'm implementing a statistics calculator that processes grade lists to compute key academic metrics including average score, pass/excellent rates and counts using efficient Python comprehensions and mathematical operations for quick educational assessment.

```
Click to add a breakpoint sults(grades, average, pass_rate, excellent_rate, pass_count, excellent_count):
37      print("\n=== Grade Analysis Results ===")
38      print(f"Total students: {len(grades)}")
39      print(f"Average grade: {average:.2f}")
40      print(f"Passing students: {pass_count}")
41      print(f"Excellent students: {excellent_count}")
42      print(f"Pass rate: {pass_rate:.2f}%")
43      print(f"Excellent rate: {excellent_rate:.2f}%")
44
45      display_all_grades(grades)
46
47  def display_all_grades(grades):
48      print(f"\nAll grades: {', '.join(map(str, grades))}")
```

I'm creating result display functions that present a comprehensive grade analysis report with formatted statistics including student counts, averages, pass/excellent metrics, and a complete list of all individual grades for full data transparency.

```
50  def main():
51      grades = input_grades()
52
53      if not grades:
54          print("No grades entered.")
55          return
56
57      average, pass_rate, excellent_rate, pass_count, excellent_count = calculate_statistics(grades)
58      display_results(grades, average, pass_rate, excellent_rate, pass_count, excellent_count)
59
60  if __name__ == "__main__":
61      main()
```

I'm implementing the main program controller that coordinates the complete workflow: collecting grades, checking for valid input, computing statistics through function calls, and displaying the final analysis report, and creating an integrated grade processing system.

```
=== Grade Analysis Results ===
Total students: 4
Average grade: 44.00
Passing students: 2
Excellent students: 0
Pass rate: 50.00%
Excellent rate: 0.00%

All grades: 33, 22, 55, 66
>>>
```

SUCCESSFUL ! ! !

C++ Work:

```
5   // Structure to hold all statistics
6   struct GradeStatistics
7   {
8       double average;
9       double passRate;
10      double excellentRate;
11      int passCount;
12      int excellentCount;
13      int maxGrade;
14      int minGrade;
15      int totalStudents;
16  };
```

I'm creating a C++ struct to store comprehensive grade statistics with member variables for average scores, pass/excellent rates and counts, min/max grades, and total students, establishing a structured data container for academic performance analysis.

```cpp
vector<int> inputGrades()
{
    vector<int> grades;
    int score;
    int count = 0;

    printf("Enter student grades (max 20, enter -1 to finish):\n");

    while (count < 20)
    {
        printf("Enter grade #%d: ", count + 1);

        if (scanf("%d", &score) != 1)
        {
            printf("Invalid input. Please enter a number.\n");
            while (getchar() != '\n');
        }
        int score
        Generate Copilot summary
        if (score == -1)
        {
            break;
        }

        if (score < 0 || score > 100)
        {
            printf("Grade must be between 0-100. Please try again.\n");
            continue;
        }

        grades.push_back(score);
        count++;
    }

    return grades;
}
```

I'm implementing a C++ grade input function using vectors that collects up to 20 student grades with robust validation, handling invalid inputs through scanf error checking, range validation (0-100), and providing clear user guidance with input buffer clearing for reliable data collection.

```cpp
GradeStatistics calculateStatistics(const vector<int>& grades)
{
    GradeStatistics stats;
    int sum = 0;
    stats.passCount = 0;
    stats.excellentCount = 0;
    stats.maxGrade = grades[0];
    stats.minGrade = grades[0];
    stats.totalStudents = grades.size();

    for (int grade : grades)
    {
        sum += grade;

        // Update max and min
        if (grade > stats.maxGrade)
        {
            stats.maxGrade = grade;
        }
        if (grade < stats.minGrade)
        {
            stats.minGrade = grade;
        }

        // Count passing and excellent grades
        if (grade >= 50)
        {
            stats.passCount++;
        }
        if (grade >= 80)
        {
            stats.excellentCount++;
        }
    }

    stats.average = static_cast<double>(sum) / grades.size();
    stats.passRate = (static_cast<double>(stats.passCount) / grades.size()) * 100;
    stats.excellentRate = (static_cast<double>(stats.excellentCount) / grades.size()) * 100;

    return stats;
}
```

I'm doing the process of computing sum, min/max grades, pass/excellent counts, and then deriving average and rates with proper type casting for accurate floating-point results, efficiently populating the GradeStatistics structure.

```cpp
void displayResults(const vector<int>& grades, const GradeStatistics& stats)
{
    printf("\n=== Grade Analysis Results ===\n");
    printf("Total students: %d\n", stats.totalStudents);
    printf("Average grade: %.2f\n", stats.average);
    printf("Highest grade: %d\n", stats.maxGrade);
    printf("Lowest grade: %d\n", stats.minGrade);
    printf("Passing students: %d\n", stats.passCount);
    printf("Excellent students: %d\n", stats.excellentCount);
    printf("Pass rate: %.2f%%\n", stats.passRate);
    printf("Excellent rate: %.2f%%\n", stats.excellentRate);

    displayAllGrades(grades);
}

void displayAllGrades(const vector<int>& grades)
{
    printf("\nAll grades: ");
    for (size_t i = 0; i < grades.size(); i++)
    {
        printf("%d", grades[i]);
        if (i < grades.size() - 1)
        {
            printf(", ");
        }
    }
    printf("\n");
}
```

I'm creating result display functions that present a formatted grade analysis report showing all calculated statistics with proper formatting, followed by a comma-separated list of all individual grades for complete data visibility.

```cpp
int main()
{
    vector<int> grades = inputGrades();

    if (grades.empty())
    {
        printf("No grades entered.\n");
        return 0;
    }

    GradeStatistics stats = calculateStatistics(grades);
    displayResults(grades, stats);

    return 0;
}
```

I'm implementing the main program workflow that coordinates the complete grade processing system: collecting input data, validating for empty results, computing comprehensive statistics, and displaying the final analysis report.

```
=== Grade Analysis Results ===
Total students: 20
Average grade: 31.65
Passing students: 1
Excellent students: 0
Pass rate: 5.00%
Excellent rate: 0.00%

All grades: 10, 20, 50, 23, 32, 36, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33

ziyue@XMAS MINGW64 /c/users/ziyue/documents/code2/p9work
$
```

SUCCESSFUL！！！

Side by Side Comparison

```cpp
// Structure to hold all statistics
struct GradeStatistics
{
    double average;
    double passRate;
    double excellentRate;
    int passCount;
    int excellentCount;
    int maxGrade;
    int minGrade;
    int totalStudents;
};
```

```python
return average, pass_rate, excellent_rate, pass_count, excellent_count
```

C++'s struct provides compile-time type safety and explicit memory layout, ensuring better performance and reliability for data-intensive operations, though it requires more verbose code with explicit type declarations. Python's approach using multiple return values offers greater flexibility and conciseness, eliminating the need for formal structure definitions, but sacrifices type safety and may lead to runtime errors that could have been caught during compilation in C++.

```c
if (scanf("%d", &score) != 1)
{
    printf("Invalid input. Please enter a number.\n");
    while (getchar() != '\n');
    continue;
}
```

```python
def input_grades():
    grades = []

    print("Enter student grades (max 20, enter -1 to finish):")

    count = 0
    while count < 20:
        user_input = input(f"Enter grade #{count + 1}: ")

        if user_input == '-1':
            break

        if user_input.isdigit():
            score = int(user_input)
            if 0 <= score <= 100:
                grades.append(score)
                count += 1
            else:
                print("Grade must be between 0-100. Please try again.")
        else:
            print("Please enter a valid integer.")

    return grades
```

C++'s scanf provides formatted input with type specification, offering precise control over input parsing. Python's input method combined with string validation functions like isdigit() creates more intuitive and beginner-friendly code, though it lacks the fine-grained type control and may require additional exception handling for robust operation.

```c
    stats.average = static_cast<double>(sum) / grades.size();
    stats.passRate = (static_cast<double>(stats.passCount) / grades.size()) * 100;
    stats.excellentRate = (static_cast<double>(stats.excellentCount) / grades.size()) * 100;

    return stats;
}
```

```python
def calculate_statistics(grades):
    total = sum(grades)
    pass_count = sum(1 for score in grades if score >= 50)
    excellent_count = sum(1 for score in grades if score >= 80)

    average = total / len(grades)
    pass_rate = (pass_count / len(grades)) * 100
    excellent_rate = (excellent_count / len(grades)) * 100

    return average, pass_rate, excellent_rate, pass_count, excellent_count

```

C++ demonstrates superior performance in statistical calculations by processing all required metrics in a single pass through the data, minimizing memory access and computational overhead. Python's functional approach using built-in functions like sum() and generator expressions results in more readable and expressive code, but often requires multiple iterations through the data, potentially impacting performance with large datasets.

```cpp
void displayResults(const vector<int>& grades, const GradeStatistics& stats)
{
    printf("\n=== Grade Analysis Results ===\n");
    printf("Total students: %d\n", stats.totalStudents);
    printf("Average grade: %.2f\n", stats.average);
    printf("Highest grade: %d\n", stats.maxGrade);
    printf("Lowest grade: %d\n", stats.minGrade);
    printf("Passing students: %d\n", stats.passCount);
    printf("Excellent students: %d\n", stats.excellentCount);
    printf("Pass rate: %.2f%%\n", stats.passRate);
    printf("Excellent rate: %.2f%%\n", stats.excellentRate);

    displayAllGrades(grades);
}
```

```python
def display_results(grades, average, pass_rate, excellent_rate, pass_count, excellent_count):
    print("\n=== Grade Analysis Results ===")
    print(f"Total students: {len(grades)}")
    print(f"Average grade: {average:.2f}")
    print(f"Passing students: {pass_count}")
    print(f"Excellent students: {excellent_count}")
    print(f"Pass rate: {pass_rate:.2f}%")
    print(f"Excellent rate: {excellent_rate:.2f}%")

    display_all_grades(grades)
```

C++'s printf offers powerful and type-safe formatting capabilities with precise control over output presentation, though its syntax can be complex and error-prone for beginners. Python's f-strings provide intuitive and highly readable string interpolation that integrates seamlessly with variable names, making code more maintainable while sacrificing some of the advanced formatting options available in C++.

C++ functions with explicit type signatures enable comprehensive compile-time checking, catching type mismatches early in the development process and providing clear interfaces for large-scale projects. Python's dynamic function definitions allow for rapid prototyping and more flexible code organization, but defer type checking to runtime, potentially allowing subtle bugs to persist until execution.

C++ excels in performance optimization and type safety through its compile-time checking and explicit memory management, making it ideal for performance-critical applications where resource control and reliability are paramount. However, this comes with increased code complexity, verbose syntax, and a steeper learning curve. Python prioritizes developer productivity and code readability with its concise syntax, dynamic typing, and rich built-in functions, enabling rapid prototyping and reduced development time, though it sacrifices runtime performance and defers error detection to execution. Ultimately, C++ offers precision and speed at the cost of development efficiency, while Python delivers simplicity and agility while accepting performance trade-offs.

Reflection

What is the most important thing you learned from this and why?

The most important lesson was understanding how different programming languages represent distinct trade-offs between performance and productivity, which helps in selecting the right tool for specific projects rather than seeking a one-size-fits-all solution.

How quickly were you able to learn the basics of Python?

Python's syntax proved remarkably intuitive, allowing me to grasp fundamental concepts like lists and functions within hours, largely due to its clean, English-like structure that minimizes cognitive load.

What were the trickiest parts of getting started with Python?

The most challenging aspect was adapting to Python's dynamic typing and indentation-based syntax after C++'s explicit type declarations and brace-based blocks, requiring careful attention to spacing and variable types.

Where and how did your understanding of the programming concepts help you with this?

My existing knowledge of core programming concepts like loops, functions, and data structures transferred seamlessly between languages, allowing me to focus on syntax differences rather than learning fundamentals from scratch.

References

Generative AI acknowledgement

I used AI to polish my expressions.

This file has additional line breaks applied by OnTrack because they contain lines longer than the configured limit. Lines over 1000 characters long have been truncated to limit PDF page count. The orginal submission can be retrieved via the "Download Uploaded Files" function.

```python
def input_grades():
    grades = []

    print("Enter student grades (max 20, enter -1 to finish):")

    count = 0
    while count < 20:
        user_input = input(f"Enter grade #{count + 1}: ")

        if user_input == '-1':
            break

        if user_input.isdigit():
            score = int(user_input)
            if 0 <= score <= 100:
                grades.append(score)
                count += 1
            else:
                print("Grade must be between 0-100. Please try again.")
        else:
            print("Please enter a valid integer.")

    return grades

def calculate_statistics(grades):
    total = sum(grades)
    pass_count = sum(1 for score in grades if score >= 50)
    excellent_count = sum(1 for score in grades if score >= 80)

    average = total / len(grades)
    pass_rate = (pass_count / len(grades)) * 100
    excellent_rate = (excellent_count / len(grades)) * 100

    return average, pass_rate, excellent_rate, pass_count, excellent_count

def display_results(grades, average, pass_rate, excellent_rate, pass_count,
↪   excellent_count):
    print("\n=== Grade Analysis Results ===")
    print(f"Total students: {len(grades)}")
    print(f"Average grade: {average:.2f}")
    print(f"Passing students: {pass_count}")
    print(f"Excellent students: {excellent_count}")
    print(f"Pass rate: {pass_rate:.2f}%")
    print(f"Excellent rate: {excellent_rate:.2f}%")

    display_all_grades(grades)

def display_all_grades(grades):
    print(f"\nAll grades: {', '.join(map(str, grades))}")

```

```
50 def main():
51     grades = input_grades()
52 
53     if not grades:
54         print("No grades entered.")
55         return
56 
57     average, pass_rate, excellent_rate, pass_count, excellent_count =
       ↪  calculate_statistics(grades)
58     display_results(grades, average, pass_rate, excellent_rate, pass_count,
       ↪  excellent_count)
59 
60 if __name__ == "__main__":
61     main()
```

27 Test 2

Programming is a foundational skill in computing, and it is important to understand this well before progressing to other units. The tests provide a means of you demonstrating you have achieved the required knowledge and skills.

Date	Author	Comment
2025/08/06 20:25	Ziyue Meng	Planned date adjusted to 19 Sep.
2025/09/18 15:37	Ziyue Meng	Ready for Feedback
2025/09/24 13:41	Ganesh Krishnasamy	Redo
2025/10/16 13:01	Ganesh Krishnasamy	Fix and Resubmit
2025/10/17 19:58	Ganesh Krishnasamy	Complete